36th International Symposium on Algorithms and Computation

ISAAC 2025, December 7–10, 2025, Tainan, Taiwan

Edited by
Ho-Lin Chen
Wing-Kai Hon
Meng-Tsung Tsai

LIPIcs – Vol. 359 – ISAAC 2025 www.dagstuhl.de/lipics

Editors

Ho-Lin Chen
National Taiwan University, Taipei, Taiwan
holinchen@ntu.edu.tw

Wing-Kai Hon
National Tsing Hua University, Hsinchu, Taiwan
wkhon@cs.nthu.edu.tw

Meng-Tsung Tsai
Academia Sinica, Taipei, Taiwan
mttsai@iis.sinica.edu.tw

ACM Classification 2012
Theory of computation; Mathematics of computing

ISBN 978-3-95977-408-6

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/978-3-95977-408-6.

Publication date
November, 2025

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

License
This work is licensed under a Creative Commons Attribution 4.0 International license (CC-BY 4.0): https://creativecommons.org/licenses/by/4.0/legalcode.
In brief, this license authorizes each and everybody to share (to copy, distribute and transmit) the work under the following conditions, without impairing or restricting the authors' moral rights:
- Attribution: The work must be attributed to its authors.

The copyright is retained by the corresponding authors.

Digital Object Identifier: 10.4230/LIPIcs.ISAAC.2025.0

ISBN 978-3-95977-408-6 **ISSN 1868-8969** **https://www.dagstuhl.de/lipics**

LIPIcs – Leibniz International Proceedings in Informatics

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

Editorial Board

- Christel Baier (TU Dresden, DE)
- Roberto Di Cosmo (Inria and Université Paris Cité, FR)
- Faith Ellen (University of Toronto, CA)
- Javier Esparza (TU München, DE)
- Holger Hermanns (Universität des Saarlandes, Saarbrücken, DE and Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Wadern, DE)
- Daniel Král' (Leipzig University, DE and Max Planck Institute for Mathematics in the Sciences, Leipzig, DE)
- Sławomir Lasota (University of Warsaw, PL)
- Meena Mahajan (Institute of Mathematical Sciences, Chennai, IN – Chair)
- Chih-Hao Luke Ong (Nanyang Technological University, SG)
- Eva Rotenberg (Technical University of Denmark, Lyngby, DK)
- Pierre Senellart (ENS, Université PSL, Paris, France)
- Alexandra Silva (Cornell University, Ithaca, US)

ISSN 1868-8969

https://www.dagstuhl.de/lipics

Contents

Preface
Ho-Lin Chen, Wing-Kai Hon, and Meng-Tsung Tsai 0:ix

Program Committee
..... 0:xi–0:xii

External Reviewers
..... 0:xiii–0:xv

Invited Talks

Challenges and Opportunities of Graph-Based Algorithms for Similarity Search
Piotr Indyk 1:1–1:1

Hash Functions Bridging the Gap from Theory to Practice
Mikkel Thorup 2:1–2:1

Regular Papers

Circle-Segment Intersection Queries in Connected Geometric Graphs
Peyman Afshani, Yannick Bosch, and Sabine Storandt 3:1–3:16

Finding d-Cuts in Claw-Free Graphs
Jungho Ahn, Tala Eagling-Vose, Felicia Lucke, Daniël Paulusma, and Siani Smith 4:1–4:15

Graph Coloring Below Guarantees via Co-Triangle Packing
Shyan Akmal and Tomohiro Koana 5:1–5:16

A Dimension-Reducing Fréchet Simplification Oracle
Boris Aronov, Tsuri Farhana, Matthew J. Katz, and Indu Ramesh 6:1–6:20

Polynomial Equivalence of Extended Chemical Reaction Models
Divya Bajaj, Jose-Luis Castellanos, Ryan Knobel, Austin Luchsinger, Aiden Massie, Adrian Salinas, Pablo Santos, Ramiro Santos, Robert Schweller, and Tim Wylie . 7:1–7:21

Parameterized Algorithms for the Drone Delivery Problem
Simon Bartlmae, Andreas Hene, Joshua Könen, and Heiko Röglin 8:1–8:16

Small Space Encoding and Recognition of $\boldsymbol{k}$-Palindromic Prefixes
Gabriel Bathie, Jonas Ellert, and Tatiana Starikovskaya 9:1–9:16

New Algorithmic Directions in Optimal Transport and Applications for Product Spaces
Salman Beigi, Omid Etesami, Mohammad Mahmoody, and Amir Najafi 10:1–10:21

Finding Diverse Solutions in Combinatorial Problems with a Distributive Lattice Structure
Mark de Berg, Andrés López Martínez, and Frits Spieksma 11:1–11:19

Star-Based Separators for Intersection Graphs of c-Colored Pseudo-Segments
Mark de Berg, Bart M. P. Jansen, and Jeroen S. K. Lamme 12:1–12:16

36th International Symposium on Algorithms and Computation (ISAAC 2025).
Editors: Ho-Lin Chen, Wing-Kai Hon, and Meng-Tsung Tsai
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Optimal Online Bipartite Matching in Degree-2 Graphs
Amey Bhangale, Arghya Chakraborty, and Prahladh Harsha 13:1–13:19

On the (In)Approximability of the Monitoring Edge Geodetic Set Problem
Davide Bilò, Giordano Colli, Luca Forlizzi, and Stefano Leucci 14:1–14:19

Parameterized Complexity of Directed Traveling Salesman Problem
Václav Blažej, Andreas Emil Feldmann, Foivos Fioravantes, Paweł Rzążewski, and Ondřej Suchý 15:1–15:18

A Dichotomy for 1-Planarity with Restricted Crossing Types Parameterized by Treewidth
Sergio Cabello, Alexander Dobler, Gašper Fijavž, Thekla Hamm, and Mirko H. Wagner 16:1–16:15

Realization of Temporally Connected Graphs Based on Degree Sequences
Arnaud Casteigts, Michelle Döring, and Nils Morawietz 17:1–17:18

An Optimal Algorithm for the Stacker Crane Problem on Fixed Topologies
Yike Chen, Ke Shi, and Chao Xu 18:1–18:12

Traffic-Oblivious Multi-Commodity Flow Network Design
Markus Chimani and Max Ilsen 19:1–19:17

Sparse Induced Subgraphs in $\boldsymbol{P_7}$-Free Graphs of Bounded Clique Number
Maria Chudnovsky, Jadwiga Czyżewska, Kacper Kluk, Marcin Pilipczuk, and Paweł Rzążewski 20:1–20:16

Covering Weighted Points Using Unit Squares
Chaeyoon Chung, Jaegun Lee, and Hee-Kap Ahn 21:1–21:14

Minimum Partition of Polygons Under Width and Cut Constraints
Jaehoon Chung, Kazuo Iwama, Chung-Shou Liao, and Hee-Kap Ahn 22:1–22:20

Precoloring Extension with Demands on Paths
Arun Kumar Das, Michal Opler, and Tomáš Valla 23:1–23:15

Enumerating the Irreducible Closed Sets of an Acyclic Implicational Base of Bounded Degree
Oscar Defrain, Arthur Ohana, and Simon Vilmin 24:1–24:15

Structural Parameterizations of Simultaneous Planarity
Thomas Depian, Simon D. Fink, Alexander Firbas, Robert Ganian, Matthias Pfretzschner, and Ignaz Rutter 25:1–25:17

A Tight Lower Bound for Online Service with Deadlines and Lazy Server
Yann Disser and Linda Thelen 26:1–26:17

Simple, Strict, Proper, and Directed: Comparing Reachability in Directed and Undirected Temporal Graphs
Michelle Döring 27:1–27:21

Pathfinding in Self-Deleting Graphs
Michal Dvořák, Dušan Knop, Michal Opler, Jan Pokorný, Ondřej Suchý, and Krisztina Szilágyi 28:1–28:15

Space-Efficient Depth-First Search via Augmented Succinct Graph Encodings
Michael Elberfeld, Frank Kammer, and Johannes Meintrup 29:1–29:16

Incidences Between Curves and Points on the Grid
Esther Ezra and Micha Sharir 30:1–30:20

Compressibility Measures and Succinct Data Structures for Piecewise Linear Approximations
Paolo Ferragina and Filippo Lari 31:1–31:15

Time-Optimal k-Server
Fabian Frei, Dennis Komm, Moritz Stocker, and Philip Whittington 32:1–32:17

Coloring Reconfiguration Under Color Swapping
Janosch Fuchs, Rin Saito, Tatsuhiro Suga, Takahiro Suzuki, and Yuma Tamura .. 33:1–33:21

Deterministic Local Problems in Radio Networks: On the Impact of Local Domination and a Bit of Advice
Pawel Garncarek, Tomasz Jurdzinski, Dariusz R. Kowalski, Shay Kutten, and Miguel A. Mosteiro 34:1–34:20

Anonymous Self-Stabilising Localisation via Spatial Population Protocols
Leszek Gąsieniec, Łukasz Kuszner, Ehsan Latif, Ramviyas Parasuraman, Paul Spirakis, and Grzegorz Stachowiak 35:1–35:17

Streaming Periodicity with Mismatches, Wildcards, and Edits
Taha El Ghazi and Tatiana Starikovskaya 36:1–36:20

Parallel Joinable B-Trees in the Fork-Join I/O Model
Michael T. Goodrich, Yan Gu, Ryuto Kitagawa, and Yihan Sun 37:1–37:21

Structural Parameters for Steiner Orientation
Tesshu Hanaka, Michael Lampis, Nikolaos Melissinos, Edouard Nemery, Hirotaka Ono, and Manolis Vasilakis 38:1–38:14

Reachability of Independent Sets and Vertex Covers Under Extended Reconfiguration Rules
Shuichi Hirahara, Naoto Ohsaka, Tatsuhiro Suga, Akira Suzuki, Yuma Tamura, and Xiao Zhou 39:1–39:20

Minimum Sum Coloring with Bundles in Trees and Bipartite Graphs
Takehiro Ito, Naonori Kakimura, Naoyuki Kamiyama, Yusuke Kobayashi, and Yoshio Okamoto 40:1–40:14

A General Framework for Finding Diverse Solutions via Network Flow and Its Applications
Yuni Iwamasa, Tomoki Matsuda, Shunya Morihira, and Hanna Sumita 41:1–41:15

Parameterized Reunion with Achromatic Number
Satyabrata Jana, Souvik Saha, Saket Saurabh, and Anannya Upasana 42:1–42:20

New Approximate Distance Oracles and Their Applications
Avi Kadria and Liam Roditty 43:1–43:17

Approximation Schemes for k-Subset Sum Ratio and k-Way Number Partitioning Ratio
Sotiris Kanellopoulos, Giorgos Mitropoulos, Antonis Antonopoulos, Nikos Leonardos, Aris Pagourtzis, Christos Pergaminelis, Stavros Petsalakis, and Kanellos Tsitouras 44:1–44:22

BFS and Reverse Shortest Paths for Ball Intersection Graphs in Three and Higher Dimensions
Matthew J. Katz, Rachel Saban, and Micha Sharir 45:1–45:15

Realizing Metric Spaces with Convex Obstacles
Sándor Kisfaludi-Bak and Leonidas Theocharous 46:1–46:21

Hardness and Fixed Parameter Tractability for Pinwheel Scheduling Problems
Yusuke Kobayashi and Bingkai Lin 47:1–47:15

Quadratic Kernel for Cliques or Trees Vertex Deletion
Soh Kumabe 48:1–48:15

Maximizing Social Welfare Among EF1 Allocations at the Presence of Two Types of Agents
Jiaxuan Ma, Yong Chen, Guangting Chen, Mingyang Gong, Guohui Lin, and An Zhang 49:1–49:19

Matchgate Signatures Under Variable Permutations
Boning Meng and Yicheng Pan 50:1–50:18

Distributed Complexity of P_k-Freeness: Decision and Certification
Masayuki Miyamoto 51:1–51:21

Local Routing on Ordered Θ-Graphs
André van Renssen and Shuei Sakaguchi 52:1–52:14

A Parameterized Study of Secluded Structures in Directed Graphs
Jonas Schmidt, Shaily Verma, and Nadym Mallek 53:1–53:21

Reforming an Unfair Allocation by Exchanging Goods
Sheung Man Yuen, Ayumi Igarashi, Naoyuki Kamiyama, and Warut Suksompong 54:1–54:21

Preface

This volume contains the papers presented at the 36th International Symposium on Algorithms and Computation (ISAAC 2025), which was held in Tainan, Taiwan on December 8–10, 2025, organized by National Cheng Kung University. ISAAC 2025 provided a forum for researchers working in the areas of algorithms, theory of computation, and computational complexity. We received 178 submissions in response to the call for papers. Each submission received at least three reviews. Each submission coauthored by a PC member received an additional review. The program committee held electronic meetings using HotCRP. A total of 52 submissions were selected for presentation at the symposium.

The program committee selected the following papers as the recipients of ISAAC 2025 Best Paper and Best Student Paper Awards.

- **Best Paper:** New Approximate Distance Oracles and Their Applications by Avi Kadria and Liam Roditty
- **Best Student Paper:** Simple, Strict, Proper, and Directed: Comparing Reachability in Directed and Undirected Temporal Graphs by Michelle Döring

The symposium included two invited presentations, delivered by Piotr Indyk (Massachusetts Institute of Technology) and Mikkel Thorup (University of Copenhagen). Abstracts of their talks are included in this volume. On the day preceding the symposium, Academia Sinica hosted a one-day workshop titled Frontiers of Algorithms and Computation for early-arriving ISAAC attendees.

We would like to acknowledge the contributions of the following:

- **Organizer:** National Cheng Kung University
- **Co-organizer:** Academia Sinica
- **Sponsors:** National Science and Technology Council (Taiwan) and Association for Algorithms and Computation Theory (Taiwan)

We thank all authors who submitted extended abstracts for consideration, the program committee members for their scholarly efforts, and the external reviewers who assisted in the evaluation process.

December 2025 Ho-Lin Chen, Wing-Kai Hon, and Meng-Tsung Tsai

36th International Symposium on Algorithms and Computation (ISAAC 2025).
Editors: Ho-Lin Chen, Wing-Kai Hon, and Meng-Tsung Tsai
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Program Committee

- Ruben Becker (Ca' Foscari University of Venice)
- T H Hubert Chan (The University of Hong Kong)
- Hsien-Chih Chang (Dartmouth College)
- Yi-Jun Chang (National University of Singapore)
- Ho-Lin Chen (co-chair, National Taiwan University)
- Li Chen (Meta)
- Lin Chen (Zhejiang University)
- Siu-Wing Cheng (The Hong Kong University of Science and Technology)
- Nai-Hui Chia (Rice University)
- Rajesh Chitnis (University of Birmingham)
- Ran Duan (Tsinghua University)
- Leah Epstein (University of Haifa)
- Thomas Erlebach (Durham University)
- Henning Fernau (Trier University)
- Arnab Ganguly (University of Wisconsin at Whitewater)
- Prantar Ghosh (Tennessee Technological University)
- Daniel Gibney (University of Texas at Dallas)
- Mayank Goswami (City University of New York)
- Meng He (Dalhousie University)
- Shuichi Hirahara (National Institute of Informatics)
- Wing-Kai Hon (co-chair, National Tsing Hua University)
- Seok Hee Hong (University of Sydney)
- Chien-Chung Huang (ENS Ulm)
- Shang-En Huang (National Taiwan University)
- Shih-Han Hung (National Taiwan University)
- Seungbum Jo (Chungnam National University)
- Mong-Jen Kao (National Yang-Ming Chiao-Tung University)
- Ralf Klasing (CNRS and University of Bordeaux)
- Dominiki Koeppl (University of Yamanashi)
- Yi Li (Nanyang Technological University)
- Han-Hsuan Lin (National Tsing Hua University)
- Hsiang-Hsuan Liu (Utrecht University)
- Miguel A. Mosteiro (Pace University)
- Prajakta Nimbhorkar (Chennai Mathematical Institute)
- Thomas Nowak (ENS Paris-Saclay)
- Eunjin Oh (Pohang University of Science and Technology)
- Peter Rossmanith (RWTH Aachen University)

36th International Symposium on Algorithms and Computation (ISAAC 2025).
Editors: Ho-Lin Chen, Wing-Kai Hon, and Meng-Tsung Tsai
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

- Kunihiko Sadakane (The University of Tokyo)
- Saket Saurabh (Institute of Mathematical Sciences)
- Tetsuo Shibuya (The University of Tokyo)
- Hsin-Hao Su (Boston College)
- Yuichi Sudo (Hosei University)
- Meng-Tsung Tsai (co-chair, Academia Sinica)
- Seeun William Umboh (The University of Melbourne)
- Hoa Vu (San Diego State University)
- Hung-Lung Wang (National Taiwan Normal University)
- Yitong Yin (Nanjing University)
- Fang-Yi Yu (George Mason University)
- Viktor Zamaraev (University of Liverpool)
- Louxin Zhang (National University of Singapore)

External Reviewers

- Duncan Adamson
- Akanksha Agrawal
- Hee-Kap Ahn
- Sheikh Shakil Akhtar
- Marcelo Arenas
- Yossi Azar
- Hans-Joachim Böckenhauer
- Niranka Banerjee
- Sandip Banerjee
- Jeremy Barbay
- Robert Barish
- Michael Bekos
- Matthias Bentert
- Ronak Bhadra
- Subhash Bhagat
- Sougata Bose
- Yann Bourreau
- Reilly Browne
- Thai Bui
- Elisabet Burjons
- Alessio Campanelli
- Daniele Carnevale
- Paul Cesaretti
- Sankardeep Chakraborty
- Harish Chandramouleeswaran
- Sobyaachi Chatterjee
- Jiale Chen
- Jingbang Chen
- Li-Hsuan Chen
- Po-An Chen
- Xiaoyu Chen
- Hou-Teng Cheong
- Po-Yu Chou
- Rezaul Chowdhury
- Marek Chrobak
- Alexis de Colnet
- Antonio Cruciani
- Antoine Dailly
- Peter Davies-Peck
- Dipan Dey
- Sanjana Dey
- Quentin Dubroff
- David Eppstein
- Tomer Even
- Szilard Fazekas
- Weiming Feng
- Zhidan Feng
- Orr Fischer
- Henry Fleischmann
- Xinyu Fu
- Zhimeng Gao
- Leszek Gasieniec
- Matthias Gehnen
- Pratik Ghosal
- Amin Shiraz Gilani
- Eric Gottlieb
- Christoph Grüne
- Clemens Grabmayer
- Tomer Grossman
- Xinru Guo
- Manoj Gupta
- Rachana Gusain
- Gregory Gutin
- Vishwa Prakash HV
- Michel Habib
- Tuomas Hakoniemi
- Tesshu Hanaka
- Darryl Hill
- Jion Hirose
- Gary Hoppenworth
- Helal Hossen
- Jun-Ting Hsieh

36th International Symposium on Algorithms and Computation (ISAAC 2025).
Editors: Ho-Lin Chen, Wing-Kai Hon, and Meng-Tsung Tsai
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

- Haoqiang Huang
- Yufan Huang
- Jackson Huffstutler
- Ling-Ju Hung
- Tanmay Inamdar
- Yuni Iwamasa
- Taisuke Izumi
- Satyabrata Jana
- Wojciech Janczewski
- Tom Janssen
- Shaofeng H.-C. Jiang
- Ce Jin
- Kai Jin
- Oğuzhan Külekci
- Shahin Kamali
- Prahlad Narasimhan Kasthurirangan
- Matthias Kaul
- Hwi Kim
- Sung-hwan Kim
- Yonghwan Kim
- Naoki Kitamura
- Bjørn Kjos-Hanssen
- Yasuaki Kobayashi
- László Kozma
- Bob Krekelberg
- Pankaj Kumar
- Pritesh Kumar
- En-Jui Kuo
- Kevin Kurien Alex
- Kazuhiro Kurita
- Asaf Levin
- Jason Li
- Jianqiang Li
- Jiayi Lian
- Koen Ligthart
- Girija Limaye
- Honghao Lin
- Fu-Hong Liu
- Mingmou Liu
- Zhengwei Liu
- Aleksander Łukasiewicz
- Nikolai Maas
- Kevin Mann
- Yaping Mao
- Andrea Marino
- Lili Mei
- Takuya Mieno
- Lazar Milenković
- Pranabendu Misra
- Daniel Mock
- Sounak Modak
- Hendrik Molter
- William K. Moses Jr.
- Amer E. Mouawad
- Anish Mukherjee
- Dimitrios Myrisiotis
- Yuto Nakashima
- Vassilios Nakos
- Namrata Namrata
- Trung Thanh Nguyen
- Naomi Nishimura
- Alexandre Nolin
- Zeev Nutov
- Mattia Odorisio
- Neil Olver
- Ragukumar P
- Andreas Padalkin
- Dominik Pajak
- Atasi Panda
- Fahad Panolan
- GiBeom Park
- Mano Prakash Parthasarathi
- Debasish Pattanayak
- Daniel Paul-Pena
- Théo Pierron
- Daniel Puttini
- Sandeep R. B.
- Arash Rafiey

- Benjamin Raichel
- Keshav Ranjan
- Rajath Rao
- Xuandi Ren
- Yoshifumi Sakai
- Ankita Sarkar
- Jayalal Sarma
- Srinivasa Rao Satti
- Rik Sengupta
- Aditi Sethia
- Vihan Shah
- Shuai Shao
- Roohani Sharma
- Masahiro Shibata
- Takeharu Shiraga
- Arseny Shur
- Shkha Singh
- Jack Spalding-Jamieson
- Karteek Sreenivasaiah
- Adarsh Srinivasan
- Frank Stephan
- Alicia Stepin
- Moritz Stocker
- Manuel Stoeckl
- Yann Strozecki
- Vorapong Suppakitpaisarn
- Akira Suzuki
- Hoang Ta
- Gaurish Telang
- Sharma V. Thankachan
- Samuel Thomas
- Carlo Tosoni
- Zdeněk Troníček
- Brian Tsai
- Masahito Tsukahara
- Ta-Wei Tu
- Torsten Ueckerdt
- Jara Uitto
- Jihan Wang
- Philip Whittington
- Sebastian Wild
- Fei Wu
- Kaiyu Wu
- Xuan Wu
- Mingji Xia
- Yinzhan Xu
- Akito Yamamoto
- Ryosuke Yamano
- Penghui Yao
- Sorrachai Yingchareonthawornchai
- Yuki Yonemoto
- Sébastien Zeitoun
- Chihao Zhang
- Ruilong Zhang
- Tianyi Zhang
- Yiming Zhao
- Chaodong Zheng
- Houyu Zhou
- Xinglong Zhou
- Stanislav Zivny
- Janez Žerovnik

Challenges and Opportunities of Graph-Based Algorithms for Similarity Search

Piotr Indyk ✉
Massachusetts Institute of Technology, Cambridge, MA, USA

Abstract

Over the last few years, graph-based approaches to nearest neighbor search have attracted renewed interest. Algorithms such as HNSW, NSG, and DiskANN have become popular tools in practice. These algorithms are highly versatile and come with efficient implementations. At the same time, their correctness, performance guarantees, and functionality remain poorly understood. In this talk, I will discuss the challenges and opportunities presented by this class of algorithms.

2012 ACM Subject Classification Theory of computation

Keywords and phrases Similarity search

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.1

Category Invited Talk

© Piotr Indyk;
licensed under Creative Commons License CC-BY 4.0
36th International Symposium on Algorithms and Computation (ISAAC 2025).
Editors: Ho-Lin Chen, Wing-Kai Hon, and Meng-Tsung Tsai; Article No. 1; pp. 1:1–1:1

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

Hash Functions Bridging the Gap from Theory to Practice

Mikkel Thorup
University of Copenhagen, Denmark

Abstract

Randomized algorithms are often enjoyed for their simplicity, but the hash functions employed to yield the desired probabilistic guarantees are often too complicated to be practical. Hash functions are used everywhere in computing, e.g., hash tables, sketching, dimensionality reduction, sampling, and estimation. Many of these applications are relevant to Machine Learning, where we are often interested in similarity between high dimensional objects. Reducing the dimensionality is key to efficient processing. Abstractly, we like to think of hashing as fully-random hashing, assigning independent hash values to every possible key, but essentially this requires us to store the hash values for all keys, which is unrealistic for most key universes, e.g., 64-bit keys. In practice we have to settle for implementable hash functions, and often practitioners settle for implementations that are too simple in that the algorithms end up working only for sufficiently random input. However, the real world is full of structured/non-random input. The issue is severe, for simplistic hash functions will often work very well in tests with random input. Moreover, the issue is often that error events that should never happen in practice, happen with way too high probability. This does not show in a few tests, but will show up over time when you put the system into production. Over the last decade there has been major developments in simple to implement tabulation based hash functions offering strong theoretical guarantees, so as to support fundamental properties such as Chernoff bounds, Sparse Johnson-Lindenstrauss transforms, and fully-random hashing on a given set w.h.p. etc. I will discuss some of the principles of these developments and offer insights on how far we can bridge from theory (assuming fully-random hash functions) to practice (needing something that can actually implemented efficiently).

2012 ACM Subject Classification Theory of computation

Keywords and phrases Hash functions

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.2

Category Invited Talk

Funding *Mikkel Thorup*: Supported by VILLUM Foundation Grant 54451, Basic Algorithms Research Copenhagen (BARC).

© Mikkel Thorup;
licensed under Creative Commons License CC-BY 4.0

Circle-Segment Intersection Queries in Connected Geometric Graphs

Peyman Afshani
Aarhus University, Denmark

Yannick Bosch
University of Konstanz, Germany

Sabine Storandt
University of Konstanz, Germany

Abstract

In this paper, we study the problem of efficiently reporting all intersections between a given set of line segments in the plane and a query circle, focusing on the case where the segments form the edges of a connected geometric graph. While previous data structures for circle-segment intersection queries on general segment sets incur high space or query time costs, we exploit the connectivity of the input to obtain significantly improved performance. In fact, we propose a new circle-segment intersection data structure that can be constructed in $\mathcal{O}((n+C)\log^3 n)$ time and space on connected graphs with n edges and C edge crossings. It answers intersection queries in $\mathcal{O}(k\log^3 n)$ time, where k denotes the output size. Our method relies on the construction of efficient circle-graph intersection oracles as well as a novel linear-time algorithm to partition the edges of the graph into balanced, connected components, which might be of independent interest. In a proof-of-concept experimental study on real-world road networks, we show that our novel data structure also performs well in practice. Even on networks with millions of edges, the construction time is within minutes and queries are answered in a few milliseconds.

2012 ACM Subject Classification Theory of computation → Computational geometry; Theory of computation → Data structures design and analysis

Keywords and phrases Intersection data structure, Graph partitioning, Dobkin-Kirkpatrick hierarchy

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.3

Acknowledgements This work was initiated at Dagstuhl Seminar 25191 "Adaptive and Scalable Data Structures" (2025-05-04 – 2025-05-09) `https://www.dagstuhl.de/25191`. We thank Schloss Dagstuhl -– Leibniz-Zentrum für Informatik for their support.

1 Introduction

Given a set $\mathcal{S}$ of geometric objects and a query object $\mathcal{Q}$, reporting the intersections of $\mathcal{Q}$ with elements in $\mathcal{S}$ is a fundamental task in computational geometry. In this work, we focus on the scenario where $\mathcal{S}$ is a set of segments in the Euclidean plane, and $\mathcal{Q}$ is a query circle whose center and radius is revealed at query time. The goal is to preprocess $\mathcal{S}$ into a data structure that allows to report circle-segment intersections efficiently.

Notably, there are two different types of intersections that can occur between a segment $S = (p, q)$, specified by its endpoints $p, q \in \mathbb{R}^2$, and a query circle $\mathcal{Q} = (c, r)$, specified by its center $c \in \mathbb{R}^2$ and its radius $r \in \mathbb{R}_0^+$:

- **Type-(i) intersection:** This intersection occurs if $d_2(c, p) \leq r$ and $d_2(c, q) > r$, that is, one endpoint of S is inside the query circle and the other one outside of it.
- **Type-(ii) intersection:** This intersection occurs if $d_2(c, p) > r$ and $d_2(c, q) > r$ but $d_2(c, S) \leq r$, that is, both endpoints of S are outside of the query circle but part of the segment is inside the circle.

Both intersection types are visualized in Figure 1.

© Peyman Afshani, Yannick Bosch, and Sabine Storandt;
licensed under Creative Commons License CC-BY 4.0

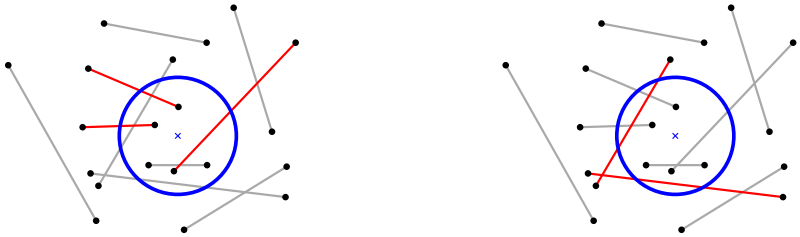


Figure 1 Circle-segment query example. Left: Segments marked red have a type-(i) intersection with the query circle. Right: Segments marked red have a type(ii)-intersection with the query circle.

Closely related to circle queries are disk queries, where segments fully contained within the disk need to be reported as well. So for disk queries, if a data structure identifies the set of segment endpoints that are contained in the disk, one can batch report all incident segments, as those are either completely contained in the disk or have a type(i)-intersection with its boundary circle. Thus, circle queries are somewhat more intricate than disk queries, as here this batch reporting cannot be applied if output-sensitive query times are desired. In [16], different data structures for each intersection type were proposed to cater for circle-segment and disk-segment intersection queries on a given set of n input segments. For fixed radius, after an $\mathcal{O}(n^{3/2}\log n)$ construction phase, a data structure of size $\mathcal{O}(n\log^2 n)$ allows to answer circle or disk queries in time $\mathcal{O}(\sqrt{n}\log^2 n + k)$ or $\mathcal{O}(\sqrt{n}\log n + k\log n)$, where k denotes the output size. For variable radius, a data structure of size $\mathcal{O}(n)$ was shown to answer queries in $\mathcal{O}(n^{2/3+\varepsilon} + k)$ for $\varepsilon > 0$. Significantly improved query times of $\mathcal{O}(\log n + k)$ are also possible but at the cost of a much higher space consumption of $\mathcal{O}(n^{3+\varepsilon})$. Other trade-offs between data structure size and query time can be obtained as well. For example, the query times for the fixed radius case can be matched for the variable radius scenario using a data structure of size $\mathcal{O}(n^{3/2+\varepsilon})$. In all these scenarios, the data structure for detecting type-(ii) intersections dominates the space consumption and the query time.

In this paper, we consider a special case of the circle- or disk-segment intersection problem, in which the segments in $\mathcal{S}$ are the edges of a connected geometric graph G. There are numerous applications of circle-edge or disk-edge intersection queries:

- **Nearest Edges.** Given a geometric graph G and a query point p, identifying the edges of G that are within distance r of p is equivalent to a disk query (p, r) on the edge segment set. This query type is useful, for example, for the construction of segment proximity graphs [1] or in edge facility location problems [20].
- **Graph parameter computation.** There are several graph parameters used to classify geometric graphs whose definition relies on the number and type of disk-segment intersections. This includes the notions of τ-lankiness [18] and λ-low-density [11].
- **Motion planning for circular robots.** Obstacle collision avoidance for robots that have to navigate a mesh or are supposed to sweep a given polygon (e.g. a floor plan) requires to detect whether the robot boundary intersects any obstacle [21]. For circular robots, a circle-segment reporting data structure can be leveraged to detect collisions.
- **Map matching.** Let G be a road network where edge costs describe travel times. A map matching query is specified by a sequence of m measurements $(p_i, r_i, t_i), i = 1, \ldots, m$ (usually obtained via GPS) where p_i denotes the measured position, r_i the error radius (stemming from measurement imprecision) and t_i the time stamp of the measurement. The goal is to match each position p_i to a location l_i in G that is within distance r_i of p_i such that l_{i+1} can be reached from l_i on a path in G with travel time at most $t_{i+1} - t_i$.

Figure 2 Exemplary circle-segment intersection queries with different radii on a connected road network. Query circles are blue and the respective intersecting segments are highlighted in red.

It was proven in [8] that a feasible map matching solution can be fully characterized by the intersection points of the circles (p_i, r_i) with edges in G, and an efficient algorithm was proposed that leverages a circle-segment data structure.

Some example queries on a city road network are visualized in Figure 2. We show that we can utilize the connectivity of G to significantly improve the construction time, the space consumption, and the query times of circle- or disk-segment intersection data structures in theory and practice.

1.1 Related Work

There are interesting special cases of the circle-segment or disk-segment intersection problem, which have been studied in the literature:

- If all input segments have length zero, the input is simply a set of points. Thus, the problem degenerates into reporting all points on the boundary of a query circle or inside a query disk. The latter is also known as the disk range searching problem. Disk range searching in $\mathbb{R}^2$ can be transformed into halfspace range searching in $\mathbb{R}^3$, for which a data structure of size $\mathcal{O}(n \log n)$ allows to answer queries in $\mathcal{O}(\log n + k)$ [5].
- If the circle has a radius of zero, it degenerates into a point. Then, the circle/disk query asks for the segments that contain the query point, also known as segment stabbing problem. If the segments are assumed to be disjoint, queries can be answered in $\mathcal{O}(\log n)$ after a $\mathcal{O}(n \log n)$ preprocessing phase that produces a data structure of linear size [12].

For the general case, the data structures for type-(i) and type-(ii) circle/disk-segment intersection described in [16] are the state-of-the-art. For variable radius, first a partition tree on the segment endpoints is constructed [19] after projecting the points to $\mathbb{R}^3$. Here, the point set is recursively partitioned using simplices, such that in the resulting hierarchical tree, the number of points in each child partition is a constant fraction of the number of points in the parent partition and only a bounded number of children intersect a query object at

each level. To only retrieve the intersecting segments with an endpoint inside the circle, the data structure is augmented such that halfplane composition queries can be answered. Also type-(ii) intersections can be identified with the help of halfplane query compositions. This gives rise to a data structure of size $\mathcal{O}(n)$ with query times in $\mathcal{O}(n^{2/3+\varepsilon} + k)$ for $\varepsilon > 0$, as well as the other trade-offs mentioned in the introduction.

For fixed radius, type-(i) and type-(ii) intersections are computed with two different data structures. For type-(i), first a stabbing path is computed on the segment endpoints. This path is guaranteed to intersect a given query circle at most $\mathcal{O}(\sqrt{n})$ times. On this path, a tree data structure is constructed, which is augmented with a halfplane query data structure at each node. Using geometric transformations and dualization, this allows to identify the segments with an endpoint outside and an endpoint inside the query circle. For type-(ii) intersections, a reduction to triangle stabbing is described and proper data structures for this use case are leveraged. In [8], the data structure for circle-segment queries with fixed radius r from [16] was refined and improved for the case in which the input segments form a path to show improved theoretical running times for map matching. However, for practical implementation, an AABB tree is used in [8], as the nested data structures described in [16] are rather intricate and expensive to construct. The AABB tree, available as a ready-to-use implementation in CGAL, is a hierarchical data structure in which each node corresponds to an axis-aligned bounding box (AABB) that encapsulates the set of geometric primitives associated with its descendant leaf nodes (in this case, the input segments). While often fast in practice, it cannot guarantee output-sensitive query times as the bounding box of a segment set might intersect a given geometric query object $\mathcal{Q}$ (as a circle) while none of the contained segments actually have an intersection with $\mathcal{Q}$.

In [4], the segment-circle intersection problem was studied, in which the roles of the input and the query are switched compared to our setting. Here, the input is a set of n circles and the query object is a segment. It was shown that the circles can be preprocessed into a data structure of size $\mathcal{O}(n \log^2 n)$ such that queries can be answered in $\mathcal{O}(n^{2/3} \log^2 n + k)$ where k denotes the output size. For disks, the query time reduces to $\mathcal{O}(\sqrt{n} \log^2 n + k)$.

There are also other contexts in which the connectivity structure of the input segments plays a crucial role. For example, in [3], the input is a set of segments and the goal is to report the connected components that intersect a given query segment. In [6], it was shown that intersections between two sets of line segments can be computed much faster if each of the two sets is connected.

1.2 Contribution

We propose a new data structure for circle-segment or disk-segment intersection queries on connected planar graphs with n edges, which can be constructed using $\mathcal{O}(n \log^3 n)$ time and space, and answers queries with output size k in time $\mathcal{O}(k \log^3 n)$. This is a significant improvement over the general case, where a data structure of (near-)linear size yields query times in $\mathcal{O}(n^{2/3+\varepsilon} + k)$ for $\varepsilon > 0$. Thus, especially for queries with small output size, our query time is vastly superior. If the input graph is non-planar and there are C crossings among the segments, the construction time and the space consumption increase to $\mathcal{O}((n + C) \log^3 n)$ but the query time stays the same. At the heart of our data structure lies a scheme which partitions the edges of the connected input graph, such that the resulting partitions have balanced size and each still form connected components. We show that such a partitioning can be computed in time $\mathcal{O}(n)$. Applying this result recursively allows us to efficiently construct

an edge partition tree[1] of logarithmic depth. We then equip each such edge partition with an oracle that decides whether for a given query circle $\mathcal{Q}$ there is some edge in the set that intersects $\mathcal{Q}$. Based on this oracle, the tree can be traversed efficiently on query time.

We also carefully describe how to implement the proposed data structure and demonstrate its usefulness in a proof-of-concept evaluation on different road networks. While road networks are non-planar, the number of crossings C between the edges is typically in $\Theta(\sqrt{n})$ [15] and thus we expect our data structure to perform well. This is confirmed in the experiments where we observe fast construction times even on large networks, and improved query times over the AABB tree implementation provided by CGAL.

2 Preliminaries

Throughout the paper, we assume to be given a connected geometric graph $G(V, E)$ as input. The graph nodes are points in $\mathbb{R}^2$ and each edge forms a straight-line segment between its endpoints. The number of pairwise crossings between edge segments is denoted by C. Here, an edge crosses another edge if their intersection contains a point that is not an endpoint of both edges. We assume that no pair of edges intersects in more than one point. Connectivity of G implies that there is a simple path between any pair of nodes in the graph, formed by sequence of edges from E. Thus, connectivity solely relies on the graph structure. This is in contrast to the notion of connected arrangement of segments, in which the set of points induced by the segments needs to be connected [3]. While in a connected geometric graph the edge segment arrangement is also connected, the reverse is not necessarily true (consider a set of segments which all pairwise cross). However, in a planar graph embedding, where we have $C = 0$, these two definitions coincide.

3 An Improved Data Structure for Connected Geometric Graphs

In this section, we introduce our new data structure that leverages the connectivity of the input graph to answer circle-edge and disk-edge intersection queries more efficiently.

In all approaches presented in [16] for the general case, the primary data structure is constructed by only considering the endpoints of the input segments. Either a spanning path with bounded stabbing number is computed on that point set or a partition tree is constructed on the (transformed) endpoints. Thus, any connectivity information is immediately lost. The original segments are only indexed in secondary or even tertiary data structures which are queried individually if the respective point sets fulfill certain requirements.

In contrast, our data structure construction is driven by the edges of G. In Section 3.1, we prove that the edges of a connected graph G can be partitioned into two subsets, such that each partition remains connected and the partition sizes are a constant fraction of the total number of edges. The respective partitioning can be computed in linear time. By applying this method recursively, we obtain an edge partition tree of logarithmic depth in which the leaves correspond to individual segments. In Section 3.2, we describe how to equip each node of the edge partition tree with an oracle that efficiently decides whether a given query circle has a type-(i) or type-(ii) intersection with some edge in the connected subgraph associated with the tree node. This allows to traverse the tree in DFS-order for query answering, only following paths for which the oracle answer is positive. More details on the query answering procedure are given in Section 3.3.

[1] We remark that despite the similar name our data structure differs substantially from the partition tree concept for point sets described in [19].

3.1 Balanced Connected Graph Partitioning

In the construction of our intersection search data structure, we follow the well-established paradigm of recursively dividing our problem into smaller subproblems until they reach a tractable size. To accomplish this while also being able to utilize the connectivity of the input graph, we require an algorithm that partitions the edge set of the input graph such that the connectivity property is maintained in each partition. Existing geometric partitioning methods (for example, median- or grid-based) are oblivious to the connectivity structure and thus are not guaranteed to produce the desired result. The method we propose does the opposite as it completely ignores all geometric information and instead operates only on the abstract graph. It is thus also applicable to non-geometric graphs. Edge partitioning has previously been studied in the context of distributed graph algorithms and parallelization [9, 22]. However, there the goal is to minimize communication cost introduced by the partitioning or to minimize node duplication (which turns out to be NP-hard). However, our focus is on balanced partitioning into connected components. In the remainder of this subsection, we prove the following central theorem.

▶ **Theorem 1.** *Given a connected graph $G[E]$ induced by the edge set E of size n, there is an $\mathcal{O}(n)$-time algorithm that partitions E into E_1 and E_2 such that:*
- $E = E_1 \uplus E_2$,
- $G[E_i]$ *is connected for* $i \in \{1, 2\}$,
- $|E_i| \leq \frac{2}{3}n$ *for* $i \in \{1, 2\}$.

We now describe the partitioning algorithm in detail and then prove its correctness to establish the theorem. The algorithm starts by constructing a spanning tree of G via a DFS run from an arbitrarily selected source node s. Let T_v denote the nodes of the subtree of the DFS-tree that is rooted at node v. The following observation sketches how we can use the notion of T_v to achieve our goal of connected edge partitioning.

▶ **Observation 2.** *Let $E(T_v)$ be the set of edges in E that are incident to nodes in T_v. The graphs induced by the edge partitions $E_1 := E(T_v)$ and $E_2 := E \setminus E_1$ are each connected.*

But we still need to take care of the size bound for each partition. Thus, we would like to efficiently compute the function $\phi(v) := |E(T_v)|$ that assigns to each node the number of edges incident to the nodes in T_v. Initially, we set $\phi(v) = 0$ for all nodes v. We then consider the nodes one after the other using a post-order traversal of the DFS-tree and update the ϕ-values of the current node v as well as its parent node $par(v)$ in the DFS-tree. We will maintain the invariant that at the moment that v is processed, all ϕ-values are less than $\frac{1}{3}n$. This condition is clearly fulfilled after the initialization phase.

Let us now consider the algorithm steps for processing a node v. As we use post-order traversal of the DFS-tree, we know that v is the last node to be processed in T_v. We first check whether $\phi(v) + deg(v) < \frac{1}{3}n$, where $deg(v)$ denotes the degree of v in $G[E]$. If that is the case, we add $deg(v)$ to $\phi(v)$ and then proceed to add the new $\phi(v)$ value to $\phi(u)$ where $u = par(v)$. If we still have $\phi(u) < \frac{1}{3}n$, we are done with v and proceed to the next node in the post-traversal order. Clearly, our invariant is maintained in this case. Otherwise, if $\phi(u) \geq \frac{1}{3}n$, we also know that $\phi(u) \leq \frac{2}{3}n$, as our invariant guaranteed that $\phi(u)$ was smaller than $\frac{1}{3}n$ prior to processing v and we only updated the parent of v as $\phi(v) < \frac{1}{3}n$ held as well. Accordingly, with $v_1, \ldots, v_s$ be the children of u that were already processed up to this point (and thus contributed to $\phi(u)$), we set $E_1 := \bigcup_{i=1}^{s} E(T_{v_i})$ and $E_2 = E \setminus E_1$. The resulting partitions are each connected, as E_1 contains all edges induced by T_u and additionally only edges with at least one endpoint in T_u, while E_2 contains all induced edges of $\{T_s \setminus T_u\} \cup \{u\}$ and additionally only edges with at least one endpoint in this node set.

If, however, $\phi(v) + deg(v) \geq \frac{1}{3}n$, we know that edges incident to nodes in T_v already suffice to form E_1. If additionally $\phi(v) + deg(v) \leq \frac{2}{3}n$, we simply return $E_1 = E(T_v)$ and $E_2 = E \setminus E_1$. If the degree of v is too large to include all of its incident edges in E_1, we only use $\frac{2}{3}n - \phi(v)$ many, giving priority to edges $\{w, v\}$ with $par(w) = v$. Note that there are less than $\frac{1}{3}n$ such prioritized edges, as otherwise $\phi(v)$ would have been equal to or larger than $\frac{1}{3}n$ already, which would contradict our invariant. Therefore, we are sure to include all edges induced by T_v and therefore, again, guarantee connectivity of the resulting partitions.

Thus, the described algorithm reliably produces edge partitions E_1 and E_2, whose induced graphs are connected. However, we did not take double counting of the edges into consideration so far. An edge $e = \{a, b\} \in E$ with $a, b \in T_v$ might contribute twice to $\phi(v)$, as both $deg(a)$ and $deg(b)$ are part of the sum. Therefore, the ϕ-values might overestimate the real number of edges in the respective subtree by a factor of up to 2. Thus, we can so far only guarantee a weaker size bound than the one promised in Theorem 1.

▶ **Lemma 3.** *The partitioning algorithm runs in $\mathcal{O}(n)$ and produces connected edge partitions E_1 and E_2 each of which has a size of at most $\frac{5}{6}n$.*

Proof. The DFS run takes $\mathcal{O}(n)$ time. In the post-order traversal, each node is processed at most once, and the processing step takes constant time if we do not return the result after this step, and at most linear time for the step after which the algorithm terminates. Therefore, the total processing time is in $\mathcal{O}(n)$. By construction, the returned partitions E_1 and E_2 are connected and the estimated size of E_1 is between $\frac{1}{3}n$ and $\frac{2}{3}n$. But as some edges might have contributed twice to this estimation, it follows that we actually have $\frac{1}{6}n \leq |E_1| \leq \frac{2}{3}n$. With $E_2 = E \setminus E_1$, it follows that $\frac{1}{3}n \leq |E_2| \leq \frac{5}{6}n$. ◀

We remark that this size bound is already sufficient for our goal of constructing a partition tree on the edge set with logarithmic depth, and the simplicity of the algorithm also makes it very suitable for practical implementation. However, we can ensure a better partition size balance within the same asymptotic running time as follows. To avoid double counting, we first refine the DFS-tree to a binary tree by introducing for all nodes u with more than two children a complete binary tree rooted at u with the children forming the tree leaves. By the property of complete binary trees the total number of nodes and edges in the DFS-tree is at most doubled by this step. Moreover, we compute a lowest-common-ancestor (LCA) data structure on the refined DFS-tree rooted at s. Such a data structure can be constructed on trees in linear time and it answers LCA queries for node pairs in $\mathcal{O}(1)$ [7]. We then use the processing algorithm described above with two modifications: When processing a dummy node introduced in the binary tree refinement step, we assume its degree to be zero. When processing a real node v, we still add $deg(v)$ to $\phi(v)$, but we additionally iterate over its neighbors w with $\{v, w\} \in E$ and decrement $\phi(LCA(v, w))$ by one if w comes later in the processing order than v. We refer to this algorithm as improved partitioning algorithm.

▶ **Lemma 4.** *The improved partitioning algorithm runs in $\mathcal{O}(n)$ and produces connected edge partitions E_1 and E_2 each of which has a size of at most $\frac{2}{3}n$.*

Proof. The binary refinement step and the LCA data structure construction both run in linear time. The processing of v now takes $\mathcal{O}(deg(v))$, as the LCA data structure needs to be queried for each neighbor. Summed over all nodes, the degrees add up to $2n$ and therefore the overall running time is still linear.

Next, we argue that after v was processed and the algorithm did not yet terminate, we have $\phi(v) = |E(T_v)|$. The proof uses induction over the nodes in post-order traversal. The first node to be processed is a leaf of the DFS-tree and thus $T_v = \{v\}$. If $deg(v) \leq \frac{1}{3}n$, we

set $\phi(v) = deg(v)$ and therefore we also have $\phi(v) = |E(T_v)|$. Let us now assume that the induction hypothesis applies to the first j nodes in the post-order traversal and consider node number $j+1$, which we refer to as u. For each child v of u in T_u, we know that v was processed prior to u and therefore $\phi(v) = |E(T_v)|$. If u has only a single child v, we have $\phi(u) = \phi(v) - |\{\{w,u\} \in E | w \in T_u\}|$ right before u is processed. This is true because v propagates its ϕ-value to its parent and for all $w \in T_u$ with $\{w,u\} \in E$ we have $LCA(w,u) = u$ an u comes later in the processing order than w. Now adding $deg(u)$ to $\phi(u)$, we get $\phi(u) = \phi(v) + |\{\{w,u\} \in E | w \notin T_u\}| = |E(T_u)|$. If u has two children v_1 and v_2, then for all edges $\{a,b\}$ with $a \in T_{v_1}$ and $b \in T_{v_2}$ we have $LCA(a,b) = u$ and thus $\phi(u)$ was decremented by 1 for each such edge. It follows that we have $\phi(u) = \phi(v_1) + \phi(v_2) - |\{\{w,u\} \in E | w \in T_u\}| - |\{a,b\} \in E | a \in T_{v_1} \wedge b \in T_{v_2}\}|$ right before u is processed. As the last term is exactly the number of edges that are counted twice, once in $\phi(v_1)$ and once in $\phi(v_2)$, we get again $\phi(u) = |E(T_u)|$ after adding $deg(u)$ to $\phi(u)$. Therefore, if we terminate while processing node u because $\phi(u) + deg(u) \geq \frac{1}{3}n$, we are now indeed guaranteed that E_1 contains the promised number of edges.

But we might also terminate after processing some node v as the ϕ-value of its parent node u exceeded the threshold after adding $\phi(v)$. We first observe that this can only happen if v is the last child of u to be processed. If it is the first child, then $\phi(u) \leq 0$ holds prior to the processing of v. The value is initialized with zero, but might even become negative it u is the LCA of the endpoints of an edge that connects nodes in the respective subtree. Therefore, after the processing of v, we have $\phi(u) \leq \phi(v)$. And as $\phi(v) < \frac{1}{3}n$ for the propagation step to happen, it follows $\phi(u) < \frac{1}{3}n$ as well. If v is the last child to be processed, we know that $\phi(u) = |E(T_u)| - \{\{u,w\} \in E | w \notin T_u\}$ by the same argumentation as above. Thus, there is no double counting occurring in this scenario and it is safe to construct E_1 as described in the original partitioning algorithm. We conclude that the improved partitioning algorithm takes linear time and guarantees $\frac{1}{3}n \leq |E_i| \leq \frac{2}{3}n$ for $i \in \{1,2\}$. ◀

This completes the proof of Theorem 1. By recursively applying the theorem to a given connected input graph G until we end up with single edges, we can derive in time $\mathcal{O}(n \log n)$ an edge partition tree of depth $\mathcal{O}(\log n)$, in which each node encodes a connected subgraph of G. Figure 3 illustrates this partitioning. Next, we discuss how to augment the resulting edge partitioning tree structure to cater for circle-segment queries.

3.2 Circle-Graph Intersection Oracles

Given a connected graph $G[E]$, our goal is to construct an efficient decision oracle that returns *true* if a given query circle $\mathcal{Q}$ has a type(i)- or a type(ii)- intersection with some edge in G and *false* otherwise.

For our oracle, we utilize the well known geometric transformation in which points $p = (x,y) \in \mathbb{R}^2$ are mapped to points $p' = (x, y, x^2 + y^2) \in \mathbb{R}^3$ by a projection function $\psi(p) = p'$, and a circle $\mathcal{Q}$ with center $c = (a,b) \in \mathbb{R}^2$ and radius r is mapped to the plane $z = a(2x - a) + b(2y - b) + r^2$ in $\mathbb{R}^3$ by a function $\rho(\mathcal{Q}) = z$ such that p is inside $\mathcal{Q}$ iff $\psi(p)$ lies below $\rho(\mathcal{Q})$ and outside $\mathcal{Q}$ iff $\psi(p)$ lies above $\rho(\mathcal{Q})$. These transformations are also applied in [16]. Let know $\Psi(E) := \{\psi(p) | \{p,q\} \in E\}$ denote the set of segment endpoints in the graph after applying the transformation via ψ. Further let $CH(\Psi(E))$ denote the convex hull of the resulting three-dimensional point set. We now show how these structures can be utilized to detect intersections of type-(i).

▶ **Lemma 5.** *Iff for a given connected graph $G[E]$ and a circle $\mathcal{Q}$ the plane $\rho(\mathcal{Q})$ intersects $CH(\Psi(E))$ there exists at least one edge $e \in E$ which has a type-(i) intersection with $\mathcal{Q}$.*

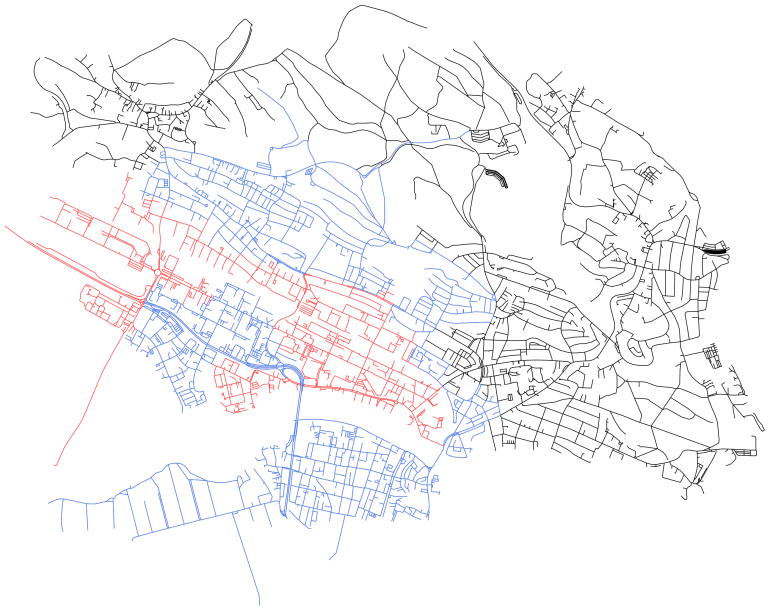


Figure 3 Partition of a graph of the road network of Konstanz with 13500 edges. The black partition represents the smaller of the two initial partitions. The blue and red partitions are the resulting connected subgraphs after subdividing the larger of the initial partitions once more.

Proof. If $\rho(\mathcal{Q})$ intersects $CH(\Psi(E))$, it follows that there is at least one point $p' \in \Psi(E)$ above $\rho(\mathcal{Q})$ as well as at least one point $q' \in \Psi(E)$ below $\rho(\mathcal{Q})$. By means of the applied geometric transformation, we know that $p = \psi^{-1}(p')$ lies outside of $\mathcal{Q}$ while $q = \psi^{-1}(q')$ lies inside of $\mathcal{Q}$. As $G[E]$ is connected, there needs to exist a path from p to q in $G[E]$, that is a path that starts outside the circle and ends in it. Accordingly, the path needs to contain at least one edge e that crosses the circle boundary. It follows that e and $\mathcal{Q}$ have a type-(i) intersection. Vice versa, if an edge that intersects $\mathcal{Q}$ exists, its endpoints are projected to different halfspaces with respect to $\rho(\mathcal{Q})$ and therefore it follows that $\rho(\mathcal{Q})$ intersects $CH(\Psi(E))$. ◀

Accordingly, the oracle returns *true* if $\rho(\mathcal{Q})$ intersects $CH(\Psi(E))$. Otherwise, if the convex hull $CH(\Psi(E))$ is either completely below or above $\rho(\mathcal{Q})$, we know that a type-(i) intersection can not occur but we still have to check for type-(ii) intersections. By definition of a type-(ii) intersection, the endpoints of the respective edge both need to be outside of the query circle. Thus, the oracle can safely return *false* if $CH(\Psi(E))$ is completely below $\rho(\mathcal{Q})$ as in this case all edge endpoints are inside the circle. So from now on, we only consider the case that $CH(\Psi(E))$ lies completely above $\rho(\mathcal{Q})$.

▶ **Lemma 6.** *Given a set of edge segments E with all segment endpoints being outside of a circle $\mathcal{Q} = (c, r)$, then there exists a type-(ii) intersection between a segment in E and $\mathcal{Q}$ iff there exists a segment $S \in E$ such that $d_2(c, S) \leq r$.*

Proof. Let $S^* \in E$ be the segment closest to the circle center c. If $d_2(c, S^*) > r$, we know that no segment in E penetrates the circle and thus there is no type-(ii) intersection. Otherwise, if $d_2(c, S^*) \leq r$ we know that S^* has a type-(ii) intersection with $\mathcal{Q}$ as both segment endpoints are outside the circle but the point on the segment closest to the circle center is inside the circle or on its boundary, which matches the definition of a type-(ii) intersection. ◀

Based on Lemma 5 and Lemma 6, we now have a full description of our decision oracle: Return *true* if $\rho(\mathcal{Q}) \cap CH(\Psi(E)) \neq \emptyset$, or, if $CH(\Psi(E))$ lies completely above $\rho(\mathcal{Q})$ and there is a segment within distance r from the circle center. Return *false* otherwise. The following theorem analyzes the construction and query time of this oracle.

▶ **Theorem 7.** *Given a connected geometric graph G with n edges and a query circle $\mathcal{Q}$, an oracle which decides in time $\mathcal{O}(\log^2 n)$ whether any graph edge intersects $\mathcal{Q}$ can be constructed in $\mathcal{O}((n+C)\log^2 n)$ time and space, where C is the number of edge crossings in G.*

Proof. For the first part of the oracle, we construct a Dobkin-Kirkpatrick hierarchy on the edge segment endpoints projected into $\mathbb{R}^3$. The Dobkin-Kirkpatrick hierarchy data structure can be computed in $\mathcal{O}(n \log^2 n)$ time and space on a set of n points and allows to answer extreme point queries in time $\mathcal{O}(\log n)$ [13, 14]. For the plane $\rho(\mathcal{Q})$, we issue two extreme point queries, one using the normal of the plane as optimization direction and one using the negated normal. The result will always be corner points of $CH(\Psi(E))$. If both points are below/above $\rho(\mathcal{Q})$, then $CH(\Psi(E))$ is fully below/above $\rho(\mathcal{Q})$. If one point is below and the other one above, we know that $\rho(\mathcal{Q}) \cap CH(\Psi(E)) \neq \emptyset$.

For the second part of the oracle, we need a data structure that reports the closest segment to a given query point (the circle center). We use a Segment Voronoi Diagram (SVD) for this purpose. It can be constructed in $\mathcal{O}((n+C)\log^2 n)$ time and space, where C is the number of edge crossings in G [17]. Nearest-segment queries can be answered in time $\mathcal{O}(\log^2 n)$. Thus, the second part of the oracle dominates the construction and query time. ◀

3.3 Query Answering

We are now ready to describe and analyze the full circle-segment data structure and the respective query answering algorithm.

Combining the insights from Theorem 1 and Theorem 7, we first construct an edge partition tree data structure, in which each node represents a connected subgraph of G, and then equip each such node with a decision oracle. To report all segment intersection with a query circle $\mathcal{Q}$, we first apply the transformation to the three-dimensional plane $\rho(\mathcal{Q})$. Then, we use the following recursive algorithm for query answering, with the initial call being to root node of our data structure: For the current data structure node, which represents some connected edge subset E', we ask the associated oracle whether there is any segment in E' that intersects $\mathcal{Q}$. If the answer is *false*, we stop. Otherwise, if the answer is *true*, we recursively call the algorithm on both children of the current node. We also stop if we reach a leaf node, which represents a single segment, and include this segment in the output set if it indeed intersects $\mathcal{Q}$. Figure 4 illustrates the query answering process on a small example graph.

▶ **Theorem 8.** *We can build a circle-segment data structure on the n edges of a connected geometric graph with C edge crossings in $\mathcal{O}((n+C)\log^3 n)$ time and space, which reports the k segments that intersect a given query circle in $\mathcal{O}(k \log^3 n)$ time.*

Proof. The construction time and the size of the data structure is dominated by the oracle computation for each subgraph represented in the edge partition tree. As the tree has logarithmic depth by virtue of Theorem 1 and the computation time is in $\mathcal{O}((n+C)\log^2 n)$ per level by virtue of Theorem 7, we get a total construction time in $\mathcal{O}((n+C)\log^3 n)$.

A query traverses the tree data structure in a DFS-like manner, aborting the search at nodes for which the oracle has concluded that there are no intersections to be detected in the respective subtree. Clearly, if the oracle answers *true* for any internal node of the

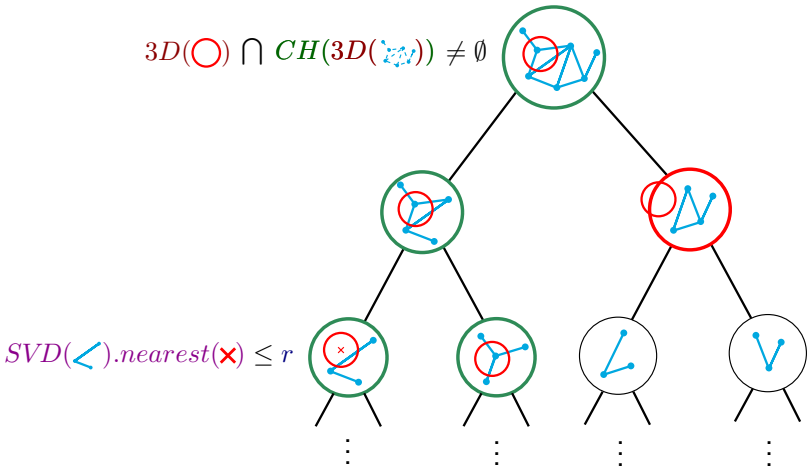


Figure 4 Circle-segment intersection data structure for a graph with 9 edges. The red circle is an exemplary query performed on the data structure. Depicted in green are nodes for which at least one of our intersection predicates (as described in Theorem 7) evaluate to true. For the red node on the right both of them return false as there are no intersecting segments.

tree, then at least for one if its children the answer needs to be *true* as well, as the segment inducing an intersection with the query circle has to be contained in either of the respective subgraphs. Therefore, the oracle calls in which the answer is *false* can always be charged to their respective siblings for which the answer is *true* (with the exception of the answer being *false* immediately on the root node). The number of times the oracle returns *true* is upper bounded by the number of nodes on a root-to-leaf path to segments that do intersect the query circle. As there are k such segments and a root-to-leaf path has a length in $\mathcal{O}(\log n)$, there are at most $\mathcal{O}(k \log n)$ such successful oracle calls. According to Theorem 7, each oracle call takes $\mathcal{O}(\log^2 n)$ time, and therefore we end up with a total query time in $\mathcal{O}(k \log^3 n)$. ◀

We remark that our data structure can also easily be used to answer disk queries with the same performance bounds. The only difference is that during query answering, whenever we encounter a node which represents a subgraph $G[E']$ for which $CH(\Psi(E'))$ lies completely below $\rho(\mathcal{Q})$, we add all edges in E' to the output set and still abort the search at this node.

4 Proof-Of-Concept Study

To demonstrate the usefulness of our circle-segment intersection data structure for practical application, we implemented the construction algorithm and the query answering routine in C++ and tested them on real-world road networks extracted from OSM. Experiments were conducted on an AMD Ryzen Threadripper 3970X with 32 cores.

4.1 Baseline

As baseline, we use the the AABB tree implementation provided by CGAL. It is a hierarchical data structure, in which each leaf node stores a geometric object and each internal node stores the bounding rectangle of all the objects in the respective subtree. It is constructed by dividing the input set of geometric objects roughly in half, computing the respective bounding rectangles, and then continuing recursively on both subsets until the leaf nodes are reached, which store the objects themselves. In our case, the leaves store the segments of the

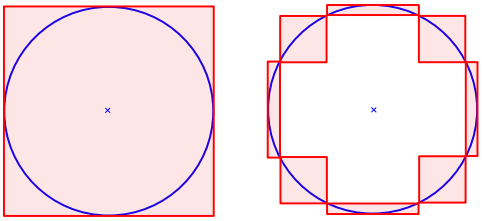


Figure 5 Query answering types using the AABB tree. Left: Simple bounding rectangle. Right: Better approximation of the circle by using eight rectangles.

input graph $G(V, E)$. In a query, given a geometric object, we check whether its bounding rectangle intersects the bounding rectangle associated with the root node of the AABB tree. If this is the case, both children are checked as well. If not, the search is aborted at the respective node. If a leaf node is reached and the final check confirms an intersection, the geometric object stored in said leaf node is added to the output. While often fast in practice, the theoretical running time can be huge in the case that the bounding rectangle of the query object intersects many bounding rectangles associated with the tree nodes, but there are no actual intersections between the query object and the elements stored in the tree leaves. This happens particularly often if the query object is not an area (as a disk) but rather an area boundary (as a circle). Here, any segment fully contained in the circle will create a bounding rectangle that intersects the one of the circle, although the segment itself does not intersect the circle. To improve the performance of our baseline, we therefore do not query the segment AABB tree with the bounding rectangle of the circle, but instead cover the circle with eight rectangles of much smaller total area, see Figure 5. We refer to this method as the *refined* AABB query.

4.2 Comparative Evaluation

As benchmarks, we used two large real-world road networks, namely a part of Germany (Figure 6) with roughly 1.1 million edges and the road network of Taiwan (Figure 8) with about 5 million edges. The Germany graph has $C = 2550$ edge crossings and the Taiwan graph $C = 22920$.

Constructing our circle-segment intersection data structure took less than 12 minutes on the Germany graph. The construction time is dominated by the segment Voronoi diagram computations for all subgraphs contained in the edge partition tree. In Figure 7, we compare the query times of our data structure to those of the refined AABB queries. We observe that except for very small query circle radii, our data structure is clearly superior. Even for large output sizes, the query times stay below 30 milliseconds and are at the peak about a factor of 8 faster than the respective refined AABB queries and two orders of magnitude faster than the naive AABB queries. Thus, we only consider the refined AABB queries in the upcoming experiments. We think that the mild increase of the query time with the output size besides the shown theoretical running time of $\mathcal{O}(k \log^3 n)$ can be explained by many paths from the root to a leaf that contain an intersecting segment share long common prefixes (while the theoretical worst-case analysis assumes them to be disjoint). Moreover, most of the occurring intersections are of type-(i). Our circle-graph oracle confirms such an intersection in time $\mathcal{O}(\log n)$ and then does not need to call the more expensive oracle for a type-(ii) intersection. Thus, in practice we have fewer and more efficient oracle calls than assumed in the theoretical analysis.

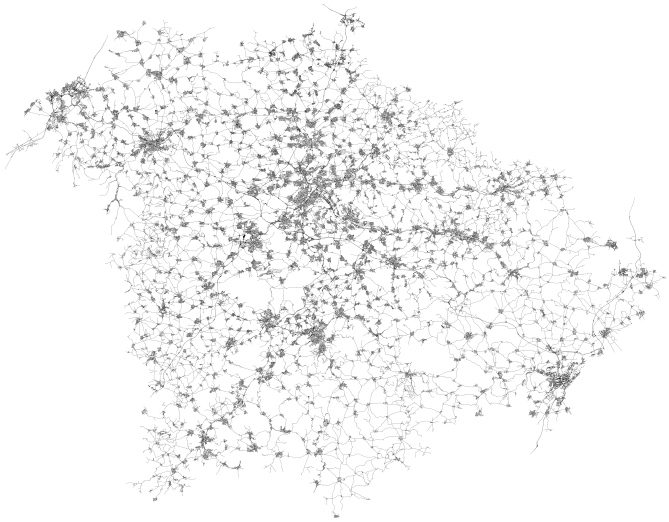


Figure 6 Road network of the metropolitan region around Stuttgart, Germany.

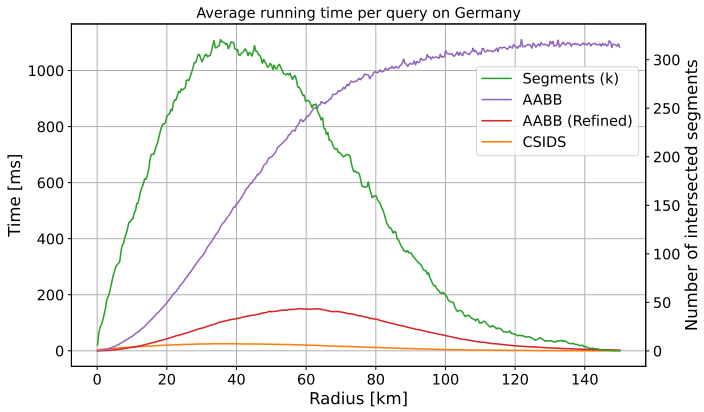


Figure 7 Query time comparison of our Circle-Segment Intersection Data Structure (CSIDS) and the AABB tree on the road network of Germany with approximately $1.1 \cdot 10^6$ edges. Query times are averaged over 100 queries per radius.

To deal with even larger graphs within a reasonable construction time, we observe that we can cut the top of our edge partition tree as well as the bottom levels. If we cut top levels, we then have to deal on query time with multiple root nodes, but we avoid the most expensive oracle construction steps on the large edge partitions close to the root. We remark that multiple root nodes can also be used in case the input graph is not connected but consists of (few) connected components. Cutting the bottom levels implies that we stop the recursive edge partitioning approach not only if we are down to single segments but already when the partition size drops below a certain threshold $t \in \mathbb{N}$. For the resulting leaf nodes, we do not construct the circle-graph intersection oracles but instead perform a simple intersection check for each contained segment with the query circle. As this limits the number of tree nodes, it helps to safe space. Furthermore, it is expected that the rather complex oracles we construct are only effective if the respective segment size is sufficiently large. We applied top and bottom cutting to the road network of Taiwan. This reduces the construction time

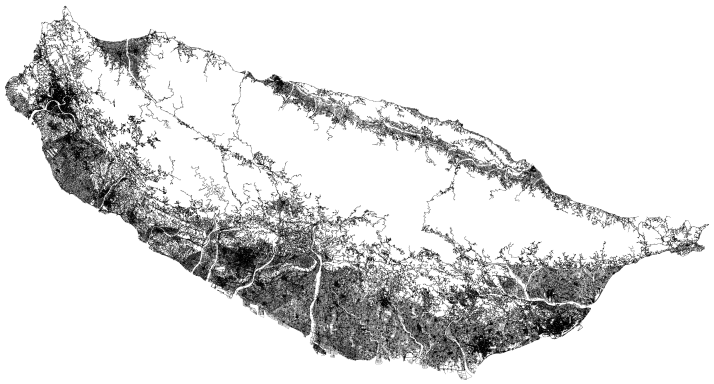


Figure 8 Road network of Taiwan.

significantly, from more than an hour to roughly 9 minutes. In Figure 9, we measure the impact of these data structure engineering techniques on the query time. Again, we observe that the refined AABB query times increase significantly with the query radius and the output size, reaching up to about half a second. Our query times, however, are again in the 20-40 millisecond range, although the search now starts from 76 root nodes (for most of which the oracle immediately returned a negative answer).

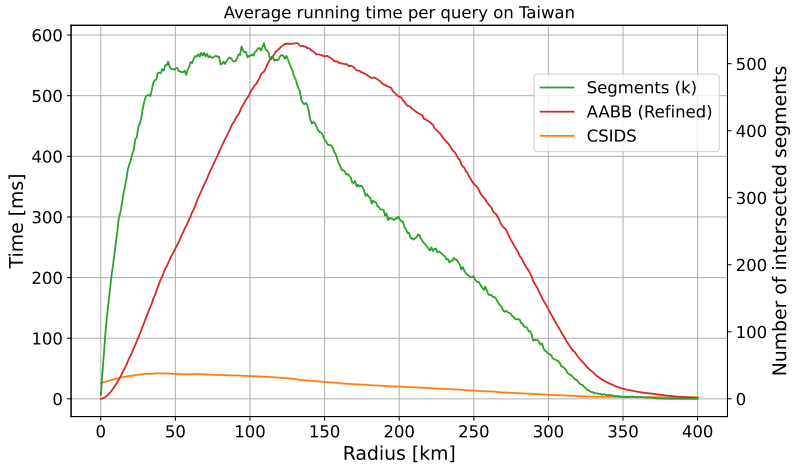


Figure 9 Query time comparison on the road network of Taiwan with approximately $5 \cdot 10^6$ edges. Our CSIDS has 76 top level query trees and the leave construction was stopped at 30 edges.

While it would of course also be interesting to conduct experiments on other types of graphs (for example, triangle meshes), our proof-of-concept study clearly shows the potential of our newly proposed data structure to accelerate the circle-segment queries on connected geometric graphs even for large edge sets.

5 Conclusions and Future Work

In this paper, we introduced a novel data structure for circle-segment intersection queries on connected geometric graphs. We proved a near-linear construction time in the size of the segment set and the number of crossings among the input segments, as well as output-sensitive query times. The data structure was also confirmed to perform well in practice. Nevertheless, there are many ways in which our data structure might be further improved:

- At the moment, we compute the circle-graph intersection oracle independently for each node in our edge partition tree. However, it might be much faster to compute the Dobkin-Kirkpatrick hierarchy and the segment Voronoi diagram bottom-up, using randomized incremental algorithms or clever merging strategies to derive the proper data structures for the level above from the already computed solutions from below. This might allow to decrease the construction time, both in theory and practice.
- Fractional cascading is often used to improve query times if the query involves repeated (binary) searches over similar or nested structures [10]. Recently, it was shown that for an input graph G with bounded degree and a set of hyperplanes in 3D associated with each node in G, one can decide for a query point and a connected subgraph H of G wether the point is below or above the lower envelope of the hyperplanes associated with the nodes in H much faster using fractional cascading [2]. Adapting this methodology to our use case, where we perform searches over nested convex hulls when we traverse the tree, could help to further improve our query times.

It would also be interesting to obtain lower bounds for data size and query time trade-offs in the connected graph setting to see how much improvement is still possible in theory. Moreover, integrating additional engineering ideas could also increase the scalability of the data structure and therefore make it useful for an even larger range of practical applications.

References

1 Ahmed Abdelkader and David M Mount. Approximate nearest-neighbor search for line segments. In *37th International Symposium on Computational Geometry*, 2021.

2 Peyman Afshani, Yakov Nekrich, and Frank Staals. Convexity helps iterated search in 3d. In *41st International Symposium on Computational Geometry (SoCG 2025)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.SoCG.2025.3.

3 Pankaj K. Agarwal and M van Kreveld. Connected component and simple polygon intersection searching. *Algorithmica*, 15(6):626–660, 1996. doi:10.1007/BF01940884.

4 Pankaj K Agarwal, Marc van Kreveld, and Mark Overmars. Intersection queries for curved objects. In *Proceedings of the seventh annual symposium on Computational geometry*, pages 41–50, 1991.

5 Alok Aggarwal, Mark Hansen, and Thomas Leighton. Solving query-retrieval problems by compacting voronoi diagrams. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 331–340, 1990.

6 Julien Basch, Leonidas J Guibas, and GD Ramkumar. Reporting red-blue intersections between two sets of connected line segments. In *Algorithms – ESA'96: Fourth Annual European Symposium Barcelona, Spain, September 25–27, 1996 Proceedings 4*, pages 302–319. Springer, 1996.

7 Michael A Bender, Martin Farach-Colton, Giridhar Pemmasani, Steven Skiena, and Pavel Sumazin. Lowest common ancestors in trees and directed acyclic graphs. *Journal of Algorithms*, 57(2):75–94, 2005. doi:10.1016/J.JALGOR.2005.08.001.

8 Yannick Bosch and Sabine Storandt. Continuous map matching to paths under travel time constraints. In *23rd International Symposium on Experimental Algorithms, SEA 2025, July 22-24, 2025, Venice, Italy*, volume 338 of *LIPIcs*, pages 7:1–7:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPICS.SEA.2025.7.

9 Florian Bourse, Marc Lelarge, and Milan Vojnovic. Balanced graph edge partition. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1456–1465, 2014. doi:10.1145/2623330.2623660.

10 Bernard Chazelle and Leonidas J Guibas. Fractional cascading: I. a data structuring technique. *Algorithmica*, 1(1):133–162, 1986. doi:10.1007/BF01840440.

11 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 Thirteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 75–83. SIAM, 2011. doi:10.1137/1.9781611972917.8.

12 Mark De Berg. *Computational geometry: algorithms and applications*. Springer Science & Business Media, 2000.

13 David P Dobkin and David G Kirkpatrick. Fast detection of polyhedral intersection. *Theoretical computer science*, 27(3):241–253, 1983. doi:10.1016/0304-3975(82)90120-7.

14 David P Dobkin and David G Kirkpatrick. Determining the separation of preprocessed polyhedra—a unified approach. In *International Colloquium on Automata, Languages, and Programming*, pages 400–413. Springer, 1990. doi:10.1007/BFB0032047.

15 David Eppstein and Michael T Goodrich. Studying (non-planar) road networks through an algorithmic lens. In *Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*, pages 1–10, 2008.

16 Prosenjit Gupta, Ravi Janardan, and Michiel Smid. On intersection searching problems involving curved objects. In *Algorithm Theory – SWAT'94: 4th Scandinavian Workshop on Algorithm Theory Aarhus, Denmark, July 6–8, 1994 Proceedings 4*, pages 183–194. Springer, 1994. doi:10.1007/3-540-58218-5_17.

17 Menelaos I Karavelas. A robust and efficient implementation for the segment voronoi diagram. In *International symposium on Voronoi diagrams in science and engineering*, volume 2004, pages 51–62, 2004.

18 Hung Le and Cuong Than. Greedy spanners in euclidean spaces admit sublinear separators. *ACM Transactions on Algorithms*, 20(3):1–30, 2024.

19 Jiří Matoušek. Efficient partition trees. In *Proceedings of the seventh annual symposium on Computational geometry*, pages 1–9, 1991.

20 Harvey J Miller. Gis and geometric representation in facility location problems. *International Journal of Geographical Information Systems*, 10(7):791–816, 1996. doi:10.1080/02693799608902110.

21 A Frank Van der Stappen and Mark H Overmars. Motion planning amidst fat obstacles. In *Proceedings of the tenth annual symposium on Computational geometry*, pages 31–40, 1994.

22 Chenzi Zhang, Fan Wei, Qin Liu, Zhihao Gavin Tang, and Zhenguo Li. Graph edge partitioning via neighborhood heuristic. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 605–614, 2017. doi:10.1145/3097983.3098033.

Finding d-Cuts in Claw-Free Graphs

Jungho Ahn
Department of Computer Engineering, Inha University, Incheon, South Korea

Tala Eagling-Vose
Department of Computer Science, Durham University, Durham, UK

Felicia Lucke
LIP, ENS Lyon, Lyon, France

Daniël Paulusma
Department of Computer Science, Durham University, Durham, UK

Siani Smith
Department of Computer Science, Loughborough University, Loughborough, UK

Abstract

The Matching Cut problem is to decide if the vertex set of a connected graph can be partitioned into two non-empty sets B and R such that the edges between B and R form a matching, that is, every vertex in B has at most one neighbour in R, and vice versa. If for some integer $d \geq 1$, we allow every vertex in B to have at most d neighbours in R, and vice versa, we obtain the more general problem d-Cut. It is known that d-Cut is NP-complete for every $d \geq 1$. However, for claw-free graphs, it is only known that d-Cut is polynomial-time solvable for $d = 1$ and NP-complete for $d \geq 3$. We resolve the missing case $d = 2$ by proving NP-completeness. This follows from our more general study, in which we also bound the maximum degree. That is, we prove that for every $d \geq 2$, d-Cut, restricted to claw-free graphs of maximum degree p, is constant-time solvable if $p \leq 2d + 1$ and NP-complete if $p \geq 2d + 3$. Moreover, in the former case, we can find a d-cut in linear time. We also show how our positive results for claw-free graphs can be generalized to $S_{1^t,\ell}$-free graphs where $S_{1^t,\ell}$ is the graph obtained from a star on $t + 2$ vertices by subdividing one of its edges exactly ℓ times.

2012 ACM Subject Classification Mathematics of computing → Graph theory; Theory of computation → Graph algorithms analysis; Theory of computation → Problems, reductions and completeness

Keywords and phrases matching cut, d-cut, claw-free, maximum degree

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.4

Related Version *Full Version*: `https://arxiv.org/abs/2505.17993`

Funding *Jungho Ahn*: supported by Leverhulme Trust Research Project Grant RPG-2024-182 and INHA UNIVERSITY Research Grant.
Felicia Lucke: supported by EPSRC Grant EP/X01357X/1 and SNSF Postdoc Mobility Grant 230578.
Daniël Paulusma: supported by Leverhulme Trust Research Project Grant RPG-2024-182 and EPSRC Grant EP/X01357X/1.

Acknowledgements The authors thank Ali Momeni and Hyunwoo Lee for providing some helpful insights about the proofs of Lemma 9 and Theorem 10, respectively.

1 Introduction

In this paper, we determine new complexity results for finding d-cuts, which form a natural generalization of matching cuts. The latter form a well-studied notion that combines two classic concepts in graph theory: matchings and edge cuts. Namely, a *matching cut* in a

© Jungho Ahn, Tala Eagling-Vose, Felicia Lucke, Daniël Paulusma, and Siani Smith;
licensed under Creative Commons License CC-BY 4.0

Figure 1 Left: a graph with a matching cut (1-cut). Right: a graph with a 3-cut but no d-cut for $d \leq 2$. Figure taken from [21].

connected graph $G = (V, E)$ is a set of edges $M \subseteq E$ that is both a *matching* (i.e., no two edges in M share an end-vertex) and an *edge cut* (i.e., V can be partitioned into two non-empty sets B and R such that M consists of all the edges between B and R). The Matching Cut problem is to decide if a connected graph has a matching cut.

Already in 1984, Chvátal [9] proved that Matching Cut is NP-complete even for $K_{1,4}$-free graphs of maximum degree 4. Here, the graph $K_{1,\ell}$ denotes the *star* on $\ell + 1$ vertices, that is, the graph with vertices $u, v_1, \ldots, v_\ell$ and edges uv_i, for $i = 1, \ldots, \ell$, and a graph is *H-free* if it does not contain H as an *induced* subgraph. In contrast, Chvátal [9] also showed that Matching Cut is polynomial-time solvable for graphs of maximum degree at most 3. In 2009, Bonsma [6] proved the same for $K_{1,3}$-free graphs (of arbitrary degree), which are also known as *claw-free* graphs. Since then a growing sequence of papers appeared in which the computational complexity of Matching Cut for various special graph classes was determined, often in a systematic way (e.g., [8, 11, 18, 22, 23]) and also from a parameterized complexity perspective (e.g. [1, 12, 13, 16, 17]) and for many closely related problem variants, such as the problems of finding perfect matching cuts [5, 15, 19], maximum matching cuts [24], minimum matching cuts [20], matching multicuts [14] and disconnected perfect matchings [7]. We refer to Section 5 for a summary of the complexity results for Matching Cut restricted to H-free graphs as part of a summary for a more general notion, namely d-cuts, the topic of our paper. 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 non-empty sets B and R such that:

- the set M is the set of all edges between B and R; and
- every vertex in B has at most d neighbours in R, and vice versa.

See Figure 1 for some examples. We note that a 1-cut is a matching cut and vice versa. For a fixed integer $d \geq 1$, the d-Cut problem is to decide if a connected graph G has a d-cut. Hence, Matching Cut and 1-Cut are the same problems.

The d-Cut problem was introduced by Gomes and Sau [13]. Apart from several parameterized complexity results (see also [2]), they showed the following two results. The first result is a structural result (shown for $d = 1$ in [9, 25]), which immediately implies the first statement of the second result (shown for $d = 1$ in [9]).

▶ **Theorem 1** (Gomes and Sau [13]). *For $d \geq 1$, every graph $G = (V, E)$ with maximum degree $\Delta(G) \leq d + 2$ and $|V| > 7$ has a d-cut, which can be found in polynomial time.*

▶ **Theorem 2** (Gomes and Sau [13]). *For $d \geq 1$, d-Cut is constant-time solvable for graphs with maximum degree $\Delta \leq d + 2$ but NP-complete for $(2d + 2)$-regular graphs (that is, graphs in which every vertex has degree $2d + 2$).*

For $d \geq 1$, the results in Theorem 2 are currently the best results for d-Cut of bounded maximum degree, so for $d \geq 2$ we have a complexity gap. Gomes and Sau [13] argued that it is unlikely that the maximum degree bound of $2d + 2$ in Theorem 2 can be lowered, as that would disprove a conjecture about the existence of so-called internal partitions of r-regular graphs for odd r; see [3].

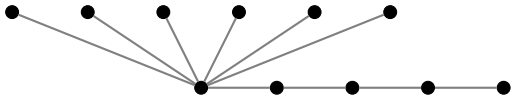


Figure 2 The graph $S_{1^6,4}$.

In particular, Theorem 2 shows that 2-Cut is constant-time solvable for graphs of maximum degree $\Delta \leq 4$ and NP-complete for graphs of maximum degree $\Delta = 6$. Gomes and Sau [13] showed that all maximum degree-5 graphs on 18 vertices without a 2-cut have some specific structure: they are 5-regular or have exactly two vertices of degree 4, which are adjacent. Interestingly, they mentioned in [13] that they were unable to find a maximum degree-5 graph on more than 18 vertices without a 2-cut.

The d-Cut problem has also been studied for H-free graphs. As mentioned, we summarize these results in Section 5. For now, we focus on the case $H = K_{1,3}$ for the following reason. It is known that for all $d \geq 3$, d-Cut is NP-complete for claw-free graphs. This was shown in [21] (the gadget constructed in the proof of Theorem 2 is not claw-free). The result for $d = 3$ contrasts the case $d = 1$ due to the aforementioned result of Bonsma [6] that 1-Cut (Matching Cut) is polynomial-time solvable on claw-free graphs and leaves open exactly the case $d = 2$. The authors of [21] could prove that 2-Cut is NP-complete for $K_{1,4}$-free graphs and asked about the case $d = 2$ in the following open problem:

▶ **Open Problem 1** (Lucke et al. [21]). *What is the complexity of* 2-Cut *for claw-free graphs?*

Our Results

Inspired by Theorems 1 and 2 of Gomes and Sau [13] and motivated by Open Problem 1, we consider *claw-free* graphs of bounded maximum degree. In Sections 3 and 4, respectively, we show the following two results for $d \geq 2$, the second of which solves Open Problem 1.

▶ **Theorem 3.** *Let $d \geq 2$. Every* claw-free *graph $G = (V, E)$ with maximum degree $\Delta(G) \leq 2d + 1$ and $|V| > 4d^2(2d + 1)$ has a d-cut, which can be found in linear time. Moreover, there exist arbitrarily large claw-free $(2d + 2)$-regular graphs with no d-cut.*

▶ **Theorem 4.** *For $d \geq 2$, d-Cut is constant-time solvable for claw-free graphs with maximum degree $\Delta \leq 2d + 1$ but* NP*-complete for claw-free graphs with maximum degree $\Delta = 2d + 3$.*

The first statement of Theorem 3 is the analogue of Theorem 1 for claw-free graphs (for which we can find the d-cut even in linear time). We show how this statement follows from a corresponding stronger statement on $S_{1^t,\ell}$-free graphs, where $S_{1^t,\ell}$ is the graph obtained from a star on $t + 2$ vertices by subdividing one of its edges exactly ℓ times, see Figure 2 for the case where $t = 6$ and $\ell = 4$. The second statement of Theorem 3 not only shows that the bound in the first statement is best possible, but also relates to Theorem 2: it shows that for every $d \geq 2$, there exists an infinite number of $(2d + 2)$-regular no-instances of d-Cut that are also claw-free.

The first statement of Theorem 3 immediately implies the constant-time part of Theorem 4, as d is a constant. We use the graph in the second statement of Theorem 3 as part of our hardness gadget in the NP-completeness part of Theorem 4.

Theorem 4 shows in particular that 2-Cut is constant-time solvable for claw-free graphs of maximum degree $\Delta \leq 5$ but NP-complete for claw-free graphs of maximum degree $\Delta = 7$, which solves Open Problem 1. Moreover, the gadget used in the reduction for d-Cut ($d \geq 3$)

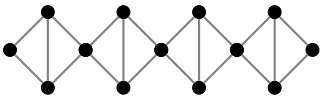


Figure 3 A chain of diamonds has no 1-cut (the diamond is the graph $K_4 - e$).

for claw-free graphs in [21] contains vertices of arbitrarily large degree. Theorem 4 strengthens this result by adding a maximum degree bound that is only one higher than the maximum degree bound in Theorem 2.

Note that d is at least 2 in both Theorems 3 and 4, which can be explained as follows. We first recall that for $d = 1$, Bonsma [6] proved that d-Cut is polynomial-time solvable even for claw-free graphs with arbitrarily large maximum degree Δ. Second, for $d = 1$, Theorem 3 would not hold: take, for example, a chain of graphs isomorphic to $K_p - e$ (complete graph minus an edge) which we obtain from k copies of $K_p - e$, for $k \geq 1$, by identifying a pair of degree $(p-2)$ vertices between consecutive copies. That is, where x_i^1, x_i^2 is the pair of degree $(p-2)$ vertices of the ith copy, we identify vertices x_i^2 and x_{i+1}^1, for all $1 \leq i \leq k-1$. The resulting chain is an arbitrarily large claw-free graph of arbitrarily large maximum degree $2p-4$ but with no 1-cut, see Figure 3 for the case where $p = 4$.

In Section 5, we give some directions for future work and show the consequence of Theorem 4 for the state-of-art summary of the complexity of d-Cut for H-free graphs.

2 Preliminaries

In this paper, we only consider finite, undirected graphs without multiple edges and self-loops. For every integer $n \geq 1$, let $[n] := \{1, \ldots, n\}$.

Let $G = (V, E)$ be a graph and let v be a vertex of G. We denote by $N_G(v) = \{u \in V \mid uv \in E\}$ the *(open) neighbourhood* of v and by $N_G[v] = N_G(v) \cup \{v\}$ the *closed neighbourhood* of v. The *degree* of v is the size of $N_G(v)$, and we denote by $\Delta(G)$ the *maximum degree* of G. For a set $S \subseteq V$, we denote by $\delta(S)$ the set of edges of G with exactly one end-vertex in S.

A *complete* graph is a graph whose vertex set is a *clique*, which is a set of pairwise adjacent vertices. For $r \geq 1$, we let K_r denote the complete graph on r vertices. A vertex set $I \subseteq V$ is an *independent set* of G if no two vertices of I are adjacent in G. In addition, G is *bipartite* if we can partition the vertex set into (possibly empty) sets A and B such that $A \cup B = V$ and A and B are each an independent set.

The *chromatic number* of a graph $G = (V, E)$ is the smallest integer k such that G has a *k-colouring*, which is a mapping $c : V \to \{1, \ldots, k\}$ with $c(u) \neq c(v)$ for every $uv \in E$. The set of all vertices that are mapped to the same colour $i \in \{1, \ldots, k\}$ is called a *colour class* of c. For $k \geq 0$, a graph G is *k-degenerate* if every subgraph of G contains a vertex of degree at most k. It is well known and readily seen that every k-degenerate graph is $(k+1)$-colourable. The *degeneracy* of G is the smallest integer k such that G is k-degenerate. The following observation holds (we give a proof for completeness).

▶ **Observation 5.** *Let $t \geq 2$ and let $G = (V, E)$ be a graph. If G has no independent set of size t, then G has a subgraph of minimum degree at least $\frac{|V|}{t-1} - 1$.*

Proof. Assume G has no independent set of size t. Let k be the degeneracy of G, so G has a subgraph of minimum degree at least k. Consider a $(k+1)$-colouring c of G. As every colour class of c is an independent set, we have $k + 1 \geq \frac{|V|}{t-1}$ and thus $k \geq \frac{|V|}{t-1} - 1$. ◀

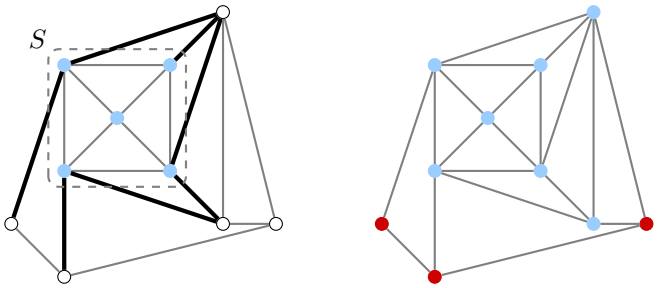


Figure 4 An example of the procedure in the proof of Lemma 9 if $d = 2$. Left: every vertex in S has at most 2 neighbours outside of S. The edges in $\delta(S)$ are highlighted in black. Right: The red-blue d-colouring we obtain from the procedure.

Let $d \geq 1$ be an integer. A *red-blue colouring* of G gives each vertex of G either colour red or blue. A *red-blue d-colouring* of G is a red-blue colouring where every vertex has at most d neighbours of the other colour and both colours, red and blue, are used at least once. See Figure 1 for an example of a red-blue 1-colouring (left figure) and a red-blue 3-colouring (right figure). In our proofs, we use the following well-known observation; see e.g. [21].

▶ **Observation 6.** *For every integer $d \geq 1$, G has a red-blue d-colouring with sets B and R if and only if the set of edges between B and R form a d-cut of G.*

Let G be a graph with a red-blue d-colouring for some $d \geq 1$. We say that a set $S \subseteq V$ is *monochromatic* if all vertices in S have the same colour.

We will also use the following observations, which can be readily seen.

▶ **Observation 7.** *Let $d \geq 1$. Let $G = (V, E)$ be a graph and let $S \subseteq V$ with $|S| \geq d+1$. For every red-blue colouring of G, in which S is monochromatic, each vertex in $V \setminus S$ with at least $d+1$ neighbours in S has the same colour as the vertices of S.*

▶ **Observation 8.** *Let $d \geq 1$. Let G be a graph with a clique K on at least $2d+1$ vertices. Then K is monochromatic in every red-blue d-colouring of G.*

3 The Proof of Theorem 3

In this section, we prove Theorem 3, which states that for all $d \geq 2$, every claw-free graph $G = (V, E)$ with maximum degree $\Delta(G) \leq 2d+1$ and $|V| > 4d^2(2d+1)$ has a d-cut, which we can find in linear time, and moreover, that there exist arbitrarily large claw-free $(2d+2)$-regular graphs with no d-cut.

We start with a lemma. In its proof, we extend the argument that Chvátal [9] used to observe that 1-Cut is constant-time solvable for graphs of maximum degree at most 3.

▶ **Lemma 9.** *Let $d \geq 1$. Let $G = (V, E)$ be a graph with $\Delta(G) \leq 2d+1$, and let S be a non-empty set of vertices with $|S| + |\delta(S)| < |V|$. If each vertex in S is incident to at most d edges in $\delta(S)$, then G has a d-cut, which can be found in linear time.*

Proof. Assume that each vertex in S is incident to at most d edges in $\delta(S)$. We will now construct a red-blue d-colouring of G, which, by Observation 6, gives that G has a d-cut. We first colour every vertex in S blue and recursively colour an uncoloured vertex blue if it has at least $d+1$ blue neighbours. Note that this takes $\mathcal{O}(|V|)$ time, as $|E| \leq (2d+1)|V|/2$. We colour all remaining vertices of G red. See also Figure 4.

Let R and B be the sets of red and blue vertices, respectively. Note that $S \subseteq B$. By construction, every vertex in R has at most d neighbours in B. As we assume that each vertex in S is incident to at most d edges in $\delta(S)$, each vertex in S has at most d neighbours in R. Since $\Delta(G) \leq 2d+1$ and each vertex in $B \setminus S$ has at least $d+1$ neighbours in B, each vertex in $B \setminus S$ has at most d neighbours in R. Hence, every vertex in B has at most d neighbours in R. It remains to show that $R \neq \emptyset$, or equivalently, that $B \neq V$, which we will do below.

Let $v_1, \ldots, v_b$ denote the vertices in $B \setminus S$ in the order they are coloured blue. For an arbitrary $i \in [b]$, let $B' := S \cup \{v_1, \ldots, v_i\}$. Recall that by construction, each vertex $v \in B' \setminus S$ has at least $d+1$ neighbours in B' and at most d neighbours not in B'. So when we colour v blue, that is, add v to B, we lose at least $d+1$ edges with exactly one blue end-vertex and gain at most d new edges with exactly one blue end-vertex. In other words, $|\delta(B' \setminus \{v\})| \geq |\delta(B')| + 1$. This implies $|\delta(S)| \geq |\delta(B)| + (|B| - |S|)$, and thus $|\delta(S)| + |S| \geq |\delta(B)| + |B|$. We use this and our assumption that $|V| > |\delta(S)| + |S|$ to deduce that

$$|V| > |\delta(S)| + |S| \geq |\delta(B)| + |B| \geq |B|,$$

and thus $B \neq V$. Hence, we obtained a red-blue d-colouring of G. ◀

To prove the first part of Theorem 3, we prove a stronger statement in Theorem 10 and afterwards we show how Theorem 10 implies the first part of Theorem 3. For every positive integer t and ℓ, we recall that $S_{1^t,\ell}$ is the graph obtained from $K_{1,t+1}$ by replacing one edge with a path of length ℓ, see Figure 2. Note that $S_{1^t,\ell}$ contains exactly $t+\ell$ edges. In addition, $S_{1^t,1}$ is the graph $K_{1,t+1}$.

▶ **Theorem 10.** *For $d \geq 2$, $t \geq 2$ and $\ell \geq 1$, every $S_{1^t,\ell}$-free graph $G = (V, E)$ with either maximum degree $\Delta = 2$, or $3 \leq \Delta \leq \frac{t}{t-1}d + \frac{1}{t-1}$ and $|V| > (d+1)\left(\frac{\Delta(\Delta-1)^{\ell+1}-2}{\Delta-2}\right)$ has a d-cut, which can be found in linear time.*

Proof. Let $d \geq 2$, $t \geq 2$ and $\ell \geq 1$. Let G be an $S_{1^t,\ell}$-free graph. If $\Delta = 2$, then G has a d-cut: take all edges between any vertex of G and its neighbours. From now on, assume $3 \leq \Delta \leq \frac{t}{t-1}d + \frac{1}{t-1}$ and and $|V| > (d+1)\left(\frac{\Delta(\Delta-1)^{\ell+1}-2}{\Delta-2}\right)$.

We choose an arbitrary vertex v of G and let $S_0 := \{v\}$. For each $i \in [\ell+1]$, we denote by S_i the set of vertices which are at distance exactly i from v in G. Note that $|S_1| \leq \Delta$ and for each $i \in [\ell]$, $|S_{i+1}| \leq (\Delta-1)|S_i|$. Thus,

$$\sum_{i=0}^{\ell+1} |S_i| \leq \frac{\Delta((\Delta-1)^{\ell+1}-1)}{\Delta-2} + 1 = \frac{\Delta(\Delta-1)^{\ell+1}-2}{\Delta-2}.$$

Let U be the set of vertices in S_ℓ which are incident to at least $d+1$ edges in $\delta(S_{\ell-1} \cup S_\ell)$. For each $u \in U$, we denote by N_u the set of neighbours of u not in $S_{\ell-1} \cup S_\ell$ and by G_u the subgraph of G induced on N_u.

We first show that N_u, for $u \in U$, has no independent set of size t. Suppose not. By the definition of S_ℓ, there is an induced path of length ℓ between v and u. Together with this path and the independent set in N_u, we can find $S_{1^t,\ell}$ as an induced subgraph, a contradiction. Thus, N_u has no independent set of size t.

For each $u \in U$, since G_u has no independent set of size t, by Observation 5, G_u has a subgraph H_u whose minimum degree is at least

$$\frac{|V(G_u)|}{t-1} - 1 \geq \frac{d+1}{t-1} - 1.$$

We remark that H_u can be found in constant time as $|N_u| \leq \Delta - 1$, which is a constant. Now, we let

$$S := \left(\bigcup_{i=0}^{\ell} S_i\right) \cup \left(\bigcup_{u \in U} V(H_u)\right).$$

Note that S can be found in constant time as $|U| \leq |S_\ell| \leq \Delta(\Delta-1)^{\ell-1}$, which is a constant.

We are going to apply Lemma 9 to S. Since $t \geq 2$, we have $t \leq 2(t-1)$, and therefore

$$\Delta \leq \frac{t}{t-1}d + \frac{1}{t-1} \leq 2d+1.$$

To apply Lemma 9 to S, we first show that every vertex in S is incident to at most d edges in $\delta(S)$. By construction, this holds for every vertex in $S \setminus \bigcup_{u \in U}(N_u \cup \{u\})$. Let u be a vertex in U and let w be a vertex in $N_u \cup \{u\}$. Note that u and w might be the same. Since the minimum degree of H_u is at least

$$\frac{d+1}{t-1} - 1$$

and u is adjacent to every vertex in N_u, there are at least

$$\frac{d+1}{t-1}$$

neighbours of w in S. Then w is incident to at most

$$\Delta - \frac{d+1}{t-1} \leq \frac{t}{t-1}d + \frac{1}{t-1} - \frac{d+1}{t-1} = d$$

edges in $\delta(S)$. That is, w is incident to at most d edges in $\delta(S)$. Hence, every vertex in S is incident to at most d edges in $\delta(S)$.

We now show that $|V| > |S| + |\delta(S)|$. Since $S \subseteq \bigcup_{i=0}^{\ell+1} S_i$, we have

$$|S| \leq \sum_{i=0}^{\ell+1} |S_i| \leq \frac{\Delta(\Delta-1)^{\ell+1} - 2}{\Delta - 2}.$$

Since every vertex in S is incident to at most d edges in $\delta(S)$, we have $|\delta(S)| \leq d|S|$. Thus,

$$|V| > (d+1)\left(\frac{\Delta(\Delta-1)^{\ell+1} - 2}{\Delta - 2}\right) \geq (d+1)|S| = |S| + d|S| \geq |S| + |\delta(S)|.$$

Hence, by Lemma 9, we can find a d-cut of G in linear time. ◀

We now show that Theorem 10 implies Theorem 3, which we recall below.

▶ **Theorem 3** (first part, restated). *For $d \geq 2$, every claw-free graph $G = (V, E)$ with maximum degree $\Delta(G) \leq 2d+1$ and $|V| > 4d^2(2d+1)$ has a d-cut, which can be found in linear time.*

Proof. Let G be a claw-free graph. If $\Delta(G) = 2$, then we can apply Theorem 10 directly. Assume $\Delta(G) \geq 3$. Theorem 10 with $t = 2$ and $\ell = 1$ implies the first part of Theorem 3. In order to see this, let G be a claw-free graph with maximum degree $\Delta(G) \leq 2d+1$ and $|V| > 4d^2(2d+1)$. As $2d+1 = \frac{t}{t-1}d + \frac{1}{t-1}$ if $t = 2$, we obtain $\Delta(G) \leq \frac{t}{t-1}d + \frac{1}{t-1}$. We also observe:

$$|V| > 4d^2(2d+1) = (2d-1)\left(\frac{(2d+1)(2d)^2}{2d-1}\right) > (d+1)\left(\frac{\Delta(\Delta-1)^2 - 2}{\Delta - 2}\right)$$

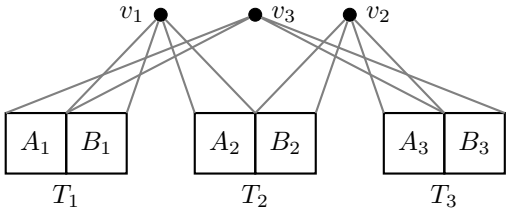


Figure 5 The graph constructed in Theorem 11 for $k = 3$. Note that for each $i \in [3]$, $A_i \cup B_i$ is a clique and v_i is complete to $B_i \cup A_{i+1}$, where $A_4 := A_1$.

where the last inequality holds from the fact that $2d - 1 \geq d + 1$ and the function $x(x-1)^2/(x-2)$ is increasing for $x \geq 3$. Hence, the conditions in Theorem 10 are satisfied, and we may conclude that G has a d-cut, which can be found in linear time. ◀

We now prove the second part of Theorem 3, which shows that the bound given in the first part of Theorem 3 is best possible. We slightly reformulate the statement as follows.

▶ **Theorem 11.** *For every integer $d \geq 2$, $k \geq 2$, and $r \geq 2d + 2$, there exists a claw-free r-regular graph on $(r + 1)k$ vertices that has no d-cut.*

Proof. Let $T_1, \ldots, T_k$ be pairwise disjoint cliques of size r. For each $i \in [k]$, let $\{A_i, B_i\}$ be a partition of T_i such that $|A_i| = d + 1$. Note that

$$|B_i| = r - (d + 1) \geq d + 1.$$

Let H be the graph obtained from $T_1 \cup \cdots \cup T_k$ by adding k new vertices $v_1, \ldots, v_k$ such that for each $i \in [k]$, $N_H(v_i)$ is equal to $B_i \cup A_{i+1}$ where $A_{k+1} := A_1$, see Figure 5. Note that H has $rk + k = (r + 1)k$ vertices.

We first show that H is claw-free and r-regular. Let i be an arbitrary integer in $[k]$. Since the neighbourhood of v_i is the disjoint union of two cliques B_i and A_{i+1}, H has no claw centred at v_i. Note further that v_i has degree r. Consider now a vertex v in T_i. The neighbourhood of v consists of all other vertices in T_i and some vertex v_j, where $j \in \{i - 1, i, k\}$. Hence, v has degree $(r - 1) + 1 = r$ and since T_i is a clique, H has no claw centred at v. We conclude that H is claw-free and r-regular.

It remains to show that H has no d-cut. Note that in any red-blue d-colouring of H, T_i, for $i \in [k]$, is monochromatic by Observation 8, since it is a clique of size at least $2d + 2$. We claim that

$$T := \bigcup_{i=1}^{k} T_i$$

is monochromatic in every red-blue d-colouring of H. Suppose for a contradiction that H has a red-blue d-colouring where T is not monochromatic. Then there exists $j \in [k]$ such that T_j and T_{j+1} have different colours, where $T_{k+1} := T_1$. Without loss of generality, assume that T_j is coloured red. Consequently, T_{j+1} is coloured blue. Since v_j has at least $d + 1$ red neighbours in B_j, it is coloured red. However, v_j has $d + 1$ blue neighbours in A_{j+1}, contradicting the fact that we considered a red-blue d-colouring. Hence, if H has a red-blue d-colouring, then T is monochromatic.

Thus, if H has a red-blue d-colouring, then T is monochromatic and hence, v_i, for $i \in [k]$, is coloured the same as T since v_i has at least $2d + 2$ neighbours in T. Therefore, there is no red-blue d-colouring of H. By Observation 6, H has no d-cut. ◀

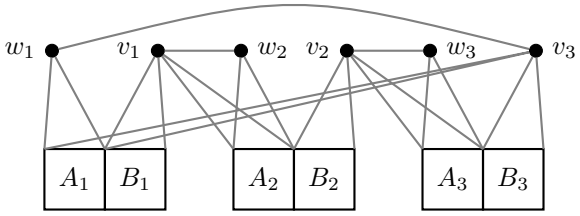


Figure 6 The graph illustrating $H_{d,k,r}$ for $k = 3$. Note that for each $i \in [3]$, $A_i \cup B_i$ is a clique, v_i is complete to $B_i \cup A_{i+1}$, where $A_4 := A_1$, and w_i is complete to A_i. If we remove w_1, w_2, and w_3, we obtain the graph in Figure 5.

4 The Proof of Theorem 4

In this section we prove Theorem 4. We first recall that the first statement of Theorem 3 immediately implies the constant-time part of Theorem 4, as d is a constant. Hence, it remains to show the NP-completeness part of Theorem 4, that is, that for every integer $d \geq 2$, d-Cut is NP-complete for claw-free graphs of maximum degree $2d + 3$. To do this, we will reduce from NAE 3-Sat 0-1, which is a variant of Not-All-Equal 3-Satisfiability (NAE 3-Sat). An instance of NAE 3-Sat consists of a set of clauses $C_1, \ldots, C_m$, each containing three variables. The problem is to decide whether there is an assignment of the variables such that every clause contains both a true and a false literal. Such an assignment is called an *NAE-satisfying* assignment.

An instance of the problem we will reduce from NAE 3-Sat 0-1 is an instance of NAE 3-Sat that is satisfied both by the assignment where every variable is true and the assignment where every variable is false. In other words, given an instance of NAE 3-Sat 0-1, every clause contains a positive literal and a negative literal. Additionally, since the clause $(\bar{x} \vee \bar{y} \vee z)$ has the same set of NAE-satisfying assignments as the clause $(x \vee y \vee \bar{z})$, we may assume that each clause contains exactly one negative literal.

The problem NAE 3-Sat 0-1 is to decide whether an instance has a third NAE-satisfying assignment. That is, we must decide whether there exists an assignment of variables such that at least one variable is true, at least one variable is false, and each clause contains both a true and a false literal. The next result was shown in [10].

▶ **Proposition 12** (Eagling-Vose et al. [10]). NAE 3-Sat 0-1 *is* NP*-complete.*

We remark that it is possible to combine the reduction from Matching Cut to NAE 3-Sat 0-1 given in [10] with the reduction given here from the latter problem to d-Cut to obtain a reduction directly from Matching Cut to d-Cut. However, we separate the two reductions for ease of explanation.

For the reduction in the proof of our NP-hardness result, we use a lemma that is based on the class of graphs from Theorem 11. For all integers $d \geq 2$, $k \geq 2$, and $r \geq 2d + 2$, let $H_{d,k,r}$ be the graph obtained from the graph in Theorem 11, with respect to the same integers, by adding k new vertices $w_1, \ldots, w_k$ such that for each $i \in [k]$, the neighbourhood of w_i in $H_{d,k,r}$ is equal to $A_i \cup \{v_{i-1}\}$ where $v_0 := v_k$. See also Figure 6.

▶ **Lemma 13.** *For all integers $d \geq 2$, $k \geq 2$, and $r \geq 2d + 2$, the graph $H_{d,k,r}$ is a claw-free graph with maximum degree $r + 1$ on $(r + 2)k$ vertices which has no d-cut. In particular, each of $w_1, \ldots, w_k$ has degree $d + 2$, and the other vertices have degree at least r.*

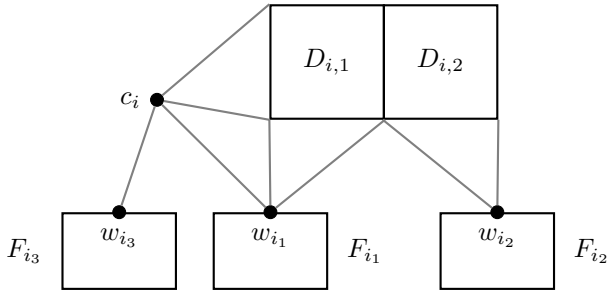


Figure 7 The clause gadget consisting of the vertex c_i and the clique $D_{i,1} \cup D_{i,2}$ representing the clause $C_i = (\overline{x}_{i_1} \vee x_{i_2} \vee x_{i_3})$, together with the corresponding variable vertices.

Proof. Let H be the graph in Theorem 11, with respect to the same integers, and let $H' := H_{d,k,r}$. Note that

$$|V(H')| = |V(H)| + k = (r+1)k + k = (r+2)k.$$

Since H is r-regular and $w_1, \ldots, w_k$ have pairwise disjoint neighbourhoods, each of $w_1, \ldots, w_k$ has degree $d+2$, and the other vertices have degree either r or $r+1$.

We show that H' is claw-free. Suppose that H' has a claw K centred at a vertex c. Since H is claw-free, K contains w_i for some $i \in [k]$. As $N_{H'}(w_i)$ is a clique, $c \neq w_i$. So $N_{H'}(c)$ is the union of two cliques, contradicting that K is a claw. Hence, H' is claw-free.

It remains to show that H' has no d-cut. By Observation 6, H' has a d-cut if and only if H' has a red-blue d-colouring. By Theorem 11, we know that $V(H)$ is monochromatic in every red-blue d-colouring. Further, w_i, for $i \in [k]$, has $d+2$ neighbours in H. Thus, by Observation 7, w_i is coloured the same as $V(H)$. This implies that $V(H')$ is monochromatic in every red-blue d-colouring and hence H' has no red-blue d-colouring. It follows that H' has no d-cut. ◀

We are now ready to prove the remaining part of Theorem 4, which we restate below.

▶ **Theorem 4** (NP-completeness part, restated)**.** *For $d \geq 2$, d-Cut is NP-complete for claw-free graphs with maximum degree $\Delta = 2d+3$.*

Proof. We reduce from NAE 3-Sat 0-1. Let ϕ be an instance of NAE 3-Sat 0-1 with variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$. In the following, we construct an instance G of d-Cut which is claw-free and has maximum degree $2d+3$. For each $\ell \in [n]$, let k_ℓ be the number of occurrences of x_ℓ in ϕ and let F_ℓ be a copy of $H_{d,k_\ell,2d+2}$. We say that a vertex of F_ℓ is *free* if it has degree $d+2$ in F_ℓ. Note that F_ℓ has exactly k_ℓ free vertices. We remark that these free vertices will play the role of variable vertices for x_ℓ.

For each clause $C_i = (\overline{x}_{i_1} \vee x_{i_2} \vee x_{i_3})$, for $i \in [m]$, we add two disjoint cliques $D_{i,1}$ of size d and $D_{i,2}$ of size $d+1$. In addition, we add a vertex c_i. We let $D_i := D_{i,1} \cup D_{i,2}$ and call $D_i \cup \{c_i\}$ a *clause gadget*. We add all edges between $D_{i,1}$ and $D_{i,2}$ and between c_i and $D_{i,1}$. For each $j \in [3]$, we choose some free vertex of F_{i_j} and call this vertex w_{i_j}. We add edges between w_{i_1} and each vertex in $D_{i,1} \cup \{c_i\}$, edges between w_{i_2} and each vertex in $D_{i,2}$, and we also add an edge between c_i and w_{i_3}, see Figure 7. We choose these free vertices in such a way that every free vertex has neighbours in exactly one clause gadget. This is possible as, for every $\ell \in [n]$, F_ℓ has k_ℓ free vertices and x_ℓ appears k_ℓ times in ϕ. Let G be the resulting graph.

We first show that G is a claw-free graph of maximum degree $2d+3$. Let ℓ be an arbitrary integer in $[n]$. Let $v \in V(F_\ell)$ be a vertex that is not free. Then, by Lemma 13, v has degree at most $2d+3$ and further, since v has neighbours only in F_ℓ, G has no claw centred at v. Let w be a free vertex of F_ℓ. Recall that w has a neighbour in a clause gadget $D_i \cup \{c_i\}$ for some $i \in [m]$. Note that w has degree at most $(d+2)+(d+1) = 2d+3$ since it has $d+2$ neighbours within F_ℓ and at most $d+1$ neighbours in $D_i \cup \{c_i\}$. In addition, the neighbours of w in $D_i \cup \{c_i\}$ form a clique of size 1 or $d+1$. Thus, the neighbourhood of w in G is the union of this clique in $D_i \cup \{c_i\}$ and some clique in F_ℓ, so G has no claw centred at w.

Finally, we consider the clause gadget $D_i \cup \{c_i\}$. Recall that c_i is a fixed vertex with neighbourhood $D_{i,1} \cup \{w_{i_1}, w_{i_3}\}$. Thus, it has degree $d+2$. Since w_{i_1} is complete to $D_{i,1}$, the neighbourhood of c_i is the union of two cliques. Hence, the vertex c_i is not the centre of a claw of G. Let u be a vertex in $D_{i,1}$. The closed neighbourhood of u is $D_i \cup \{c_i, w_{i_1}\}$, which is the union of the two cliques $D_{i,1} \cup \{c_i, w_{i_1}\}$ and D_{i_2}. Hence, u is not the centre of a claw in G. Since $|D_i| = 2d+1$, u has degree $2d+2$. Now, let u' be a vertex in $D_{i,2}$. The closed neighbourhood of u' is $D_i \cup \{w_{i_2}\}$, which is the union of the two cliques D_i and $\{w_{i_2}\}$. So u' is not the centre of a claw in G. Again, as $|D_i| = 2d+1$, the vertex u' has degree $2d+1$. It follows that G is a claw-free graph of maximum degree at most $2d+3$.

We show that G has a d-cut if and only if ϕ has an NAE-satisfying assignment with both a true and a false variable. By Observation 6, this is equivalent to showing that G has a red-blue d-colouring if and only if ϕ has an NAE-satisfying assignment with both a true and a false variable.

We assume first that G has a red-blue d-colouring and show that ϕ has an NAE-satisfying assignment with both a true and a false variable. Recall that by Lemma 13, F_ℓ is monochromatic for each $\ell \in [n]$. We set a variable x_ℓ to true if F_ℓ is coloured blue and to false otherwise.

We first show that the resulting assignment contains both a true and a false variable. Suppose for a contradiction that this is not the case. We may assume without loss of generality that all variables are true. This implies that for every $\ell \in [n]$, F_ℓ is coloured blue. Note that D_i, for $i \in [m]$, is a clique of size $2d+1$ and thus monochromatic by Observation 8. As $w_{i_2} \in F_{i_2}$, it is coloured blue. Further, w_{i_2} has $d+1$ neighbours in $D_{i,2} \subseteq D_i$. Hence, since $D_{i,2}$ is monochromatic, $D_{i,2}$ is coloured blue. It follows that $D_{i,1}$ is blue as well and, since c_i has $d+1$ blue neighbours in $D_{i,1} \cup \{w_{i_1}\}$, it is coloured blue. Thus, the whole graph is coloured blue, contradicting the fact that we considered a red-blue d-colouring. Therefore, the assignment contains both a true and a false variable.

We now show that the assignment is NAE-satisfying for ϕ. Suppose for a contradiction that the assignment is not NAE-satisfying. Then there exists a clause $C_i = (\overline{x}_{i_1} \vee x_{i_2} \vee x_{i_3})$, for some $i \in [m]$, such that all literals have the same value. Note that this implies that both x_{i_2} and x_{i_3} take the opposite value of x_{i_1}. Without loss of generality, assume that x_{i_1} is true and x_{i_2} and x_{i_3} are both false. That is, w_{i_1} is coloured blue while w_{i_2} and w_{i_3} are coloured red. Recall that D_i is monochromatic. Since w_{i_2} has $d+1$ neighbours in $D_{i,2}$, D_i is coloured the same as w_{i_2} and is thus coloured red. This implies that the blue vertex w_{i_1} has d red neighbours in $D_{i,1}$, so its neighbour c_i is coloured blue. However, as w_{i_3} is coloured red, c_i has $d+1$ red neighbours in $D_{i,1} \cup \{w_{i_3}\}$, contradicting the fact that we considered a red-blue d-colouring. Hence, the assignment is NAE-satisfying for ϕ.

For the other direction, we now assume that ϕ has an NAE-satisfying assignment with both a true and a false variable. We construct a red-blue d-colouring of G as follows. For each $\ell \in [n]$, we colour F_ℓ blue if it corresponds to a true variable and red otherwise. For each $i \in [m]$, we colour D_i the same as w_{i_2}. In addition, we colour c_i the same as w_{i_1}.

We show that this colouring is indeed a red-blue d-colouring of G. Note first that since the assignment contains both a true and a false variable, we have both red and blue vertices. It remains to show that every vertex has at most d neighbours of the other colour. Let ℓ be an arbitrary integer in $[n]$. Note that every vertex of F_ℓ which is not a free vertex has no neighbour of the other colour. Let w be a free vertex of F_ℓ. Recall that there is exactly one clause C_i, for $i \in [m]$, such that w is adjacent to the corresponding clause gadget D_i. That is, $w \in \{w_{i_1}, w_{i_2}, w_{i_3}\}$. We distinguish between these cases:

- If $w = w_{i_3}$, then it has one neighbour outside of F_ℓ and thus at most one neighbour of the other colour.
- If $w = w_{i_2}$, then, by construction, it has no neighbour of the other colour.
- If $w = w_{i_1}$, then it has $d+1$ neighbours outside of F_ℓ and, by construction, at least one of them, namely c_i, has the same colour as w.

Thus, w has at most d neighbours of the other colour.

Now, let u be a vertex in D_i. If $u \in D_{i,2}$, then its closed neighbourhood is $D_i \cup \{w_{i_2}\}$, which is monochromatic by construction. Thus, u has no neighbour of the other colour. If $u \in D_{i,1}$, then, since D_i is monochromatic, the only neighbours of u which may be coloured with the other colour are w_{i_1} and c_i. Since $d \geq 2$, it follows that u has at most d neighbours of the other colour. Thus, we may assume that $u = c_i$. Recall that the neighbourhood of c_i is $D_{i,1} \cup \{w_{i_1}, w_{i_3}\}$ and that c_i is coloured the same as w_{i_1}. Assume without loss of generality that c_i is coloured blue. Suppose for a contradiction that c_i has at least $d+1$ red neighbours. That is, $D_{i,1}$ and $w_{i,3}$ are coloured red. Since D_i is monochromatic by construction, this implies that $D_{i,2}$ and thus $w_{i,2}$ are coloured red as well. Hence, $w_{i,2}$ and $w_{i,3}$ are both coloured red, while $w_{i,1}$ is coloured blue, a contradiction to the assumption that the assignment is NAE-satisfying. Thus, c_i has at most d neighbours of the other colour. Hence, the colouring is indeed a red-blue d-colouring of G. This completes the proof. ◀

We remark that for every $\Delta > 2d+3$, d-Cut is still NP-complete on claw-free graphs with maximum degree Δ: after replacing each F_ℓ as a copy of $H_{d,k_\ell,\Delta-1}$, instead of $H_{d,k_\ell,2d+2}$, the same hardness proof works.

5 Conclusions

We proved both constant-time and NP-completeness results for d-Cut restricted to claw-free graphs of bounded maximum degree, leaving open only one case, which replaces Open Problem 1:

▶ **Open Problem 2.** *For every integer $d \geq 2$, what is the computational complexity of d-Cut on claw-free graphs of maximum degree $\Delta = 2d+2$?*

Recall that we showed in Theorem 3 that there are many claw-free $(2d+2)$-regular graphs with no d-cut and also that every claw-free graph of maximum degree $\Delta \leq 2d+1$ has a d-cut. This shows that our bound is best possible. On the other hand, our construction used in the proof of Theorem 4 for graphs of maximum degree $\Delta = 2d+3$ cannot easily be modified to work for graphs of maximum degree $\Delta \leq 2d+2$. The variable gadget is based on our construction for $(2d+2)$-regular graphs without a d-cut. We modify the construction such that it contains vertices of degree $d+2$ that can be used to connect the gadget to the rest of the construction. However, to make sure that we do not get a d-cut by partitioning the graph into a variable gadget and the rest, there has to be a vertex in the variable gadget that has at least $d+1$ neighbours outside the gadget. This leads to vertices of degree $2d+3$ in the construction. Hence, reducing the degree in this construction requires us to construct a

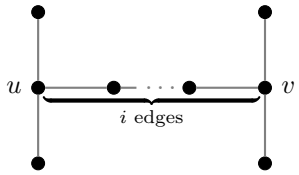


Figure 8 The graph H_i^*. Figure taken from [21].

variable gadget containing some vertices of degree smaller than $d+2$. If we want the gadget to be monochromatic, this implies we need a gadget with vertices of degree $d+1$. However, finding such a gadget seems to be a challenging task.

In our paper we also showed that our positive results for d-Cut on claw-free graphs can be generalized to even $S_{1^t,\ell}$-free graphs. A natural question, which we leave for future work, is whether our results can also be generalized to $S_{1,2,2}$-free graphs. Here, $S_{1,2,2}$ is the graph obtained from the claw by subdividing two of its edges exactly once. In order to answer this question, we need some new arguments as our current proof techniques cannot be applied immediately. For instance, in the proof of Theorem 10, a key observation is that for each $u \in U$, $G[N_u]$ has no independent set of size t, as otherwise we can find an induced $S_{1^t,\ell}$ by combining such an independent set with the path of length ℓ between u and v. This observation no longer holds for the $S_{1,2,2}$ case.

We recall that in particular, our results imply that 2-Cut, restricted to claw-free graphs of maximum degree p, is constant-time solvable if $p \leq 5$ and NP-complete if $p \geq 7$. The latter result resolved an open case in the complexity classification of d-Cut on H-free graphs, as the complexity of 2-Cut for claw-free graphs was not previously known. This brings us to the state-of-the-art summary for d-Cut on H-free graphs, which we update below.

We first give some more terminology. For $r \geq 1$, we denote by P_r and C_r, respectively, a path and a cycle on r vertices. For $s \geq 1$, we denote by sP_r the disjoint union of s copies of P_r. We denote by $S_{1,1,2}$ the graph obtained from a claw by subdividing one edge once, so that it has four edges. Let H_1^* be a graph that looks like the letter "H". It is obtained by taking the graph $2P_3$ and making the middle vertices, say u and v, of each P_3 adjacent to each other. We obtain the graph H_i^*, for $i \geq 2$, by subdividing uv exactly $i-1$ times, see Figure 8. The *girth* of a graph that is not a forest is the length of a shortest cycle in it.

We now present the updated state-of-the-art summary. For every $d \geq 1$, d-Cut is NP-complete for graphs of girth at least g for every fixed $g \geq 3$ [11], and thus for C_r-free graphs for every $r \geq 3$. It is also known that 1-Cut is NP-complete for $(H_1^*, \ldots, H_i^*)$-free graphs for every $i \geq 1$ [11], $K_{1,4}$-free graphs [9], $(3P_5, P_{15})$-free graphs [23] and $(3P_6, 2P_7, P_{14})$-free graphs [18]. On the positive side, 1-Cut is polynomial-time solvable for $(sP_3 + S_{1,1,2})$-free graphs [20], $(sP_3 + P_4 + P_6)$-free graphs [20] and $(sP_3 + P_7)$-free graphs [20] for every $s \geq 1$.

We summarize the above results and our new result for 2-Cut on claw-free graphs in the following theorem, in which all unreferenced results for d-Cut ($d \geq 2$) are shown in [21]. For two graphs H and H', we write $H \subseteq_i H'$ if H is an induced subgraph of H', and denote by $H + H'$ the disjoint union of a copy of H and a copy of H'.

▶ **Theorem 14.** *Let $d \geq 1$ and let H be a graph.*

- *If $d = 1$, then d-Cut (Matching Cut) on H-free graphs is*
  - *polynomial-time solvable if $H \subseteq_i sP_3 + S_{1,1,2}$, $sP_3 + P_4 + P_6$, or $sP_3 + P_7$ for some $s \geq 1$;*
  - *NP-complete if $H \supseteq_i K_{1,4}$, P_{14}, $2P_7$, $3P_5$, C_r for some $r \geq 3$, or H_i^* for some $i \geq 1$.*

- *If $d \geq 2$, then d-Cut on H-free graphs is*
  - *polynomial-time solvable if $H \subseteq_i sP_1 + P_3 + P_4$ or $sP_1 + P_5$ for some $s \geq 1$;*
  - NP*-complete if $H \supseteq_i K_{1,3}$, $3P_2$, C_r for some $r \geq 3$.*

It is known that for $d \geq 2$, d-Cut is polynomial-time solvable on $(H + P_1)$-free graphs whenever d-Cut is polynomial-time solvable for H-free graphs. This means that the cases $\{H + sP_1 \mid s \geq 0\}$ are all polynomially equivalent. Hence, for $d \geq 2$, there are, due to our new result, now only three non-equivalent open cases left, namely when $H \in \{2P_4, P_6, P_7\}$.

Finally, we are currently exploring the consequences of our results and proofs for finding internal partitions of graphs. To explain, a partition (V_1, V_2) of the vertex set V of a graph G is an *internal partition* of G if for every $i \in \{1, 2\}$, every vertex in V_i has at least half of its neighbours in V_i. Internal partitions are well studied; see, for example, the survey of Bazgan, Tuza and Vanderpooten [4]. The corresponding decision problem that asks whether a graph has an internal partition is known as Satisfactory Partition. As we briefly mentioned in Section 1, there are connections between d-cuts and internal partitions [13].

References

1 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. doi:10.1016/J.DAM.2021.05.016.

2 N. R. Aravind and Roopam Saxena. An FPT algorithm for Matching Cut and d-Cut. *Proc. IWOCA 2021, LNCS*, 12757:531–543, 2021. doi:10.1007/978-3-030-79987-8_37.

3 Amir Ban and Nati Linial. Internal partitions of regular graphs. *Journal of Graph Theory*, 83:5–18, 2016. doi:10.1002/JGT.21909.

4 Cristina Bazgan, Zsolt Tuza, and Daniel Vanderpooten. Satisfactory graph partition, variants, and generalizations. *European Journal of Operations Research*, 206:271–280, 2010. doi:10.1016/J.EJOR.2009.10.019.

5 Edouard Bonnet, Dibyayan Chakraborty, and Julien Duron. Cutting Barnette graphs perfectly is hard. *Theoretical Computer Science*, 1010:114701, 2024. doi:10.1016/J.TCS.2024.114701.

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. doi:10.1002/JGT.20390.

7 Valentin Bouquet and Christophe Picouleau. The complexity of the Perfect Matching-Cut problem. *Journal of Graph Theory*, 108, 2025. doi:10.1002/JGT.23167.

8 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. doi:10.1007/S00453-020-00782-8.

9 Vasek Chvátal. Recognizing decomposable graphs. *Journal of Graph Theory*, 8:51–53, 1984. doi:10.1002/JGT.3190080106.

10 Tala Eagling-Vose, Barnaby Martin, Daniel Paulusma, and Siani Smith. A forbidden subgraph study for cut problems on graphs permitting loops and multiedges. *CoRR*, abs/2502.07769, 2025. doi:10.48550/arXiv.2502.07769.

11 Carl Feghali, Felicia Lucke, Daniël Paulusma, and Bernard Ries. Matching cuts in graphs of high girth and H-free graphs. *Algorithmica*, 87:1199–1221, 2025. doi:10.1007/S00453-025-01318-8.

12 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. doi:10.1016/J.JCSS.2021.07.005.

13 Guilherme Gomes and Ignasi Sau. Finding cuts of bounded degree: complexity, FPT and exact algorithms, and kernelization. *Algorithmica*, 83:1677–1706, 2021. doi:10.1007/S00453-021-00798-8.

14 Guilherme C. M. Gomes, Emanuel Juliano, Gabriel Martins, and Vinícius Fernandes dos Santos. Matching (multi)cut: Algorithms, complexity, and enumeration. *Proc. IPEC 2024, LIPIcs*, 321:25:1–25:15, 2024. doi:10.4230/LIPICS.IPEC.2024.25.

15 Pinar Heggernes and Jan Arne Telle. Partitioning graphs into generalized dominating sets. *Nordic Journal of Computing*, 5:128–142, 1998.

16 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. doi:10.1016/J.DAM.2019.12.010.

17 Dieter Kratsch and Van Bang Le. Algorithms solving the Matching Cut problem. *Theoretical Computer Science*, 609:328–335, 2016. doi:10.1016/J.TCS.2015.10.016.

18 Hoàng-Oanh Le and Van Bang Le. Complexity results for matching cut problems in graphs without long induced paths. *Proc. WG 2023, LNCS*, 14093:417–431, 2023. doi:10.1007/978-3-031-43380-1_30.

19 Van Bang Le and Jan Arne Telle. The Perfect Matching Cut problem revisited. *Theoretical Computer Science*, 931:117–130, 2022. doi:10.1016/J.TCS.2022.07.035.

20 Felicia Lucke, Joseph Marchand, and Jannik Olbrich. Finding minimum matching cuts in H-free graphs and graphs of bounded radius and diameter. *CoRR*, abs/2502.18942, 2025. doi:10.48550/arXiv.2502.18942.

21 Felicia Lucke, Ali Momeni, Daniël Paulusma, and Siani Smith. Finding d-cuts in graphs of bounded diameter, graphs of bounded radius and H-free graphs. *Proc. WG 2024, LNCS*, 14760:415–429, 2024. full version: arXiv:2404.11389. doi:10.48550/arXiv.2404.11389.

22 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. doi:10.1016/J.TCS.2022.09.014.

23 Felicia Lucke, Daniël Paulusma, and Bernard Ries. Finding matching cuts in H-free graphs. *Algorithmica*, 85:3290–3322, 2023. doi:10.1007/S00453-023-01137-9.

24 Felicia Lucke, Daniël Paulusma, and Bernard Ries. Dichotomies for maximum matching cut: H-freeness, bounded diameter, bounded radius. *Theoretical Computer Science*, 1017:114795, 2024. doi:10.1016/J.TCS.2024.114795.

25 Augustine M. Moshi. Matching cutsets in graphs. *Journal of Graph Theory*, 13:527–536, 1989. doi:10.1002/JGT.3190130502.

Graph Coloring Below Guarantees via Co-Triangle Packing

Shyan Akmal
INSAIT, Sofia University "St. Kliment Ohridski", Bulgaria

Tomohiro Koana
Keio University, Yagami, Japan

Abstract

In the ℓ-Coloring problem, we are given a graph on n nodes, and tasked with determining if its vertices can be properly colored using ℓ colors. In this paper we study *below-guarantee graph coloring*, which tests whether an n-vertex graph can be properly colored using $g - k$ colors, where g is a trivial upper bound such as n. We introduce an algorithmic framework that builds on a packing of co-triangles $\overline{K_3}$ (independent sets of three vertices): the algorithm greedily finds co-triangles and employs a win-win analysis. If many are found, we immediately return *yes*; otherwise these co-triangles form a small *co-triangle modulator*, whose deletion makes the graph co-triangle-free.

Extending the work of [Gutin et al., SIDMA 2021], who solved ℓ-Coloring (for any ℓ) in randomized $O^*(2^k)$ time when given a $\overline{K_2}$-free modulator of size k, we show that this problem can likewise be solved in randomized $O^*(2^k)$ time when given a $\overline{K_3}$-free modulator of size k.

This result in turn yields a randomized $O^*(2^{3k/2})$ algorithm for $(n-k)$-Coloring (also known as Dual Coloring), improving the previous $O^*(4^k)$ bound. We then introduce a smaller parameterization, $(\omega + \overline{\mu} - k)$-Coloring, where ω is the clique number and $\overline{\mu}$ is the size of a maximum matching in the complement graph; since $\omega + \overline{\mu} \leq n$ for any graph, this problem is strictly harder. Using the same co-triangle-packing argument, we obtain a randomized $O^*(2^{6k})$ algorithm, establishing its fixed-parameter tractability for a smaller parameter. Complementing this finding, we show that no fixed-parameter tractable algorithm exists for $(\omega - k)$-Coloring or $(\overline{\mu} - k)$-Coloring under standard complexity assumptions.

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

Keywords and phrases coloring, parameterized algorithms, algebraic algorithms, above-guarantee, below-guarantee, subset convolution, determinants

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.5

Related Version *Full Version*: https://arxiv.org/abs/2509.12347

Funding *Shyan Akmal*: Partially funded by the Ministry of Education and Science of Bulgaria (support for INSAIT, part of the Bulgarian National Roadmap for Research Infrastructure).
Tomohiro Koana: Supported by JST ERATO Grant Number JPMJER2301, Japan.

Acknowledgements We thank the anonymous reviewers for helpful feedback on this work.

1 Introduction

Graph coloring is a cornerstone of theoretical computer science and discrete mathematics: it connects to deep structure theorems and has driven the development of numerous algorithmic techniques [3, 5, 6, 8, 14]. Given an undirected graph G on n vertices, a *proper coloring* of G is an assignment of colors to its nodes such that any two adjacent vertices receive different colors. The canonical decision problem for this concept is k-Coloring: given an integer k and a graph G, decide whether G admits a proper coloring with at most k colors.

© Shyan Akmal and Tomohiro Koana;
licensed under Creative Commons License CC-BY 4.0

While 2-Coloring is polynomial-time solvable, 3-Coloring (and therefore k-Coloring for every $k \geq 3$) is already NP-hard [18]. The field of parameterized complexity tackles such NP-hard tasks by measuring the complexity of solving these problems in terms of secondary measures. In this paper we focus on a particularly important subarea of this field known as *below-guarantee parameterization*.

Below-Guarantee Parameterization. In the guarantee parameterization paradigm, we reformulate problems by first identifying classes of instances that are trivially solvable – typically because a solution is guaranteed to exist – and then parameterize general instances by their distance from these "easy" cases. See [13] for a survey of this area.

Dual Coloring. In the context of coloring graphs with n nodes, observe that the n-Coloring problem is trivial, because we can obtain a proper coloring by assigning each vertex a distinct color. The framework of below-guarantee parameterization described above then motivates studying the $(n-k)$-Coloring problem, also referred to as Dual Coloring, where the parameter k captures how many colors we are trying to "save" compared to the trivial coloring using different colors for all vertices. The Dual Coloring problem has been influential in parameterized complexity, with its study in [7] introducing the concept of crown reductions, by now a basic technique in the field of kernelization [10, Chapter 4]. Given an n-node instance G of Dual Coloring and any constant $\varepsilon > 0$, [17] showed that in polynomial time one can kernelize the instance to a graph G' on $(2+\varepsilon)k$ vertices such that the answer to the Dual Coloring problem is the same on G and G', for sufficiently large k in terms of $\varepsilon > 0$. Combining this kernelization with existing $2^n \operatorname{poly}(n)$ time algorithms for graph coloring [5], we immediately recover a $O^*(4^{(1+\varepsilon)k})$ time algorithm for Dual Coloring for any constant $\varepsilon > 0$, where we write $O^*(f(k))$ to denote $f(k)\operatorname{poly}(n)$. Without going through this kernelization, one can also use the algorithms of [12] to solve Dual Coloring directly in $O^*(4^k)$ time. We improve upon this result:

▶ **Theorem 1.** Dual Coloring *can be solved in randomized* $O^*(2^{3k/2})$ *time.*

Let us mention a closely related problem called $(n-k)$-Set Cover. In this problem, we are given a family $\mathcal{F}$ of subsets of $[n]$, and are tasked with determining if there exist $(n-k)$ sets from $\mathcal{F}$ whose union is $[n]$. Very recently, Alferov et al. [2] showed that $(n-k)$-Set Cover can be solved in $O^*(2^{3k/2}|\mathcal{F}|)$ time. Even though the $(n-k)$-Coloring problem can be cast as an instance of $(n-k)$-Set Cover, where $\mathcal{F}$ is the collection of all independent vertex sets of the graph, this result of [2] does not imply an $O^*(2^{3k/2})$ time algorithm for $(n-k)$-Coloring, because $\mathcal{F}$ can have exponentially many sets in n. The dynamic programming approach used in the algorithm of [2] appears to inherently require this runtime dependence on $|\mathcal{F}|$ even when the sets $\mathcal{F}$ can be implicitly described as in $(n-k)$-Coloring, so that Theorem 1 is not directly implied by this previous work. However, we note it seems plausible that one could combine the ideas of [2] with the subset convolution arguments of [4] to obtain fast algorithms for Dual Coloring as well, and recover Theorem 1.

A Stronger Structural Guarantee. Given a graph G, we let $\overline{G}$ denote its complement graph, which has the same vertex set as G, but an edge $\{u,v\}$ if and only if u and v are distinct, non-adjacent nodes in G. Let $\omega = \omega(G)$ denote the size of the maximum clique in G. Let $\overline{\mu} = \mu(\overline{G})$ denote the size of the maximum matching in the complement $\overline{G}$. Then we have the following structural parameter-based guarantee for coloring G.

▶ **Observation 2.** *The graph* G *admits a proper coloring using at most* $(\omega + \overline{\mu})$ *colors.*

Proof. Let M be a maximal matching in $\overline{G}$. By definition, $|M| \leq \overline{\mu}$. By maximality of M in $\overline{G}$, the vertices that are not incident to any edge in M form an independent set in $\overline{G}$. Thus the set C of vertices outside M form a clique in G, so $|C| \leq \omega$. We color G as follows. For each edge in M, we assign a unique color to both of its endpoints, and we assign distinct new colors to all vertices in C. In total, we use at most $(\omega + \overline{\mu})$ colors. By construction this coloring is proper, which proves the claim. ◀

Note that in any graphs with n vertices, we always have $(\omega + \overline{\mu}) \leq n$. This is because if we take a clique C in G of size ω and a matching M in $\overline{G}$ of size $\overline{\mu}$, then by definition each edge of M is incident to at least one distinct vertex outside of C, which forces $n \geq |C|+|M| = (\omega + \overline{\mu})$. Thus the guarantee of Observation 2 is stronger than the trivial fact that any graph with n vertices always has a proper coloring using n colors.

In the same way this latter observation motivated the Dual Coloring problem, it is natural to ask if we can obtain efficient algorithms while parameterizing below the stronger guarantee provided by Observation 2. We prove that this is indeed possible.

▶ **Theorem 3** (Parameterization from Stronger Guarantee). *Given an integer $k \geq 1$, and a graph G that has maximum clique size ω and whose complement has maximum matching size $\overline{\mu}$, we can solve $(\omega + \overline{\mu} - k)$-Coloring in randomized $O^*(2^{6k})$ time.*

Our approach. The key ideas behind Theorems 1 and 3 are sketched below; see Section 1.1 for more details. We first greedily compute a maximal packing of co-triangles $\overline{K_3}$ (independent triplets). If the packing holds many co-triangles, saving two colors per co-triangle already yields the desired proper coloring. Otherwise, the packing's vertices form a small *co-triangle modulator* S, whose deletion results in a $\overline{K_3}$-free graph. In the latter case, we extend the randomized procedure of Gutin et al. [12], to show that we can solve the desired coloring problem in $O^*(2^{|S|})$ time.

Hardness results. We also note that algorithms which are fixed-parameter tractable with respect to k are unlikely to exist for $(\omega - k)$-Coloring and $(\overline{\mu} - k)$-Coloring, so that combining ω and $\overline{\mu}$ together in Theorem 3 appears to be necessary in order to achieve efficient parameterized algorithms with respect to these parameters. For the former, this is because 3-Coloring is NP-hard in planar graphs [11]. Since a planar graph does not have a clique of size five, this means that assuming $\mathsf{P} \neq \mathsf{NP}$ we cannot hope to solve $(\omega - k)$-Coloring in $f(k)\,\mathrm{poly}(n)$ time for any function $f(\cdot)$. To show hardness for the latter problem, we prove the following.

▶ **Theorem 4.** *The $(n/2 - k)$-Coloring problem is W[1]-hard in the parameter k on graphs with n vertices whose complement has a perfect matching.*

In view of Theorem 4, we cannot hope for an $O^*(f(k))$-time algorithm for $(\overline{\mu} - k)$-Coloring, for any function $f(\cdot)$.

1.1 Technical Overview

As a starting point, we give a brief overview for how the work of [12] implies a $O^*(4^k)$ time algorithm for Dual Coloring.

Given a host graph G and pattern graph H, we say G is *H-free* if G does not contain H as an induced subgraph. We say a set of vertices S in G is an *H-free modulator* if deleting the vertices in S from G makes the graph H-free. For $\ell \in \mathbb{N}$, we let K_ℓ denote the complete graph on ℓ vertices, and $\overline{K_\ell}$ denote its complement, the independent set on ℓ vertices.

The main relevant result from previous work is the following, proven in [12, Theorem 3.5].

▶ **Proposition 5** (Clique Modulator)**.** *Given a graph G together with a $\overline{K_2}$-free modulator of size p (i.e., a set of at most p vertices, whose deletion from G turns the graph into a clique), we can solve k-Coloring on G for any k in randomized $O^*(2^p)$ time.*

Recall that given a pattern graph H and a host graph G, we say an *H-packing* in G is a collection $\mathcal{H}$ of vertex-disjoint, induced copies of H in G. Using Proposition 5 we can solve Dual Coloring in $O^*(4^k)$ time using the following three steps.

Step 1: Identify Matching
Extract a maximal $\overline{K_2}$-packing M in G.
Step 2: Large Matching ⇒ Easy Instance
If $|M| \geq k$, then report that a proper coloring using at most $(n-k)$ colors exists.
Step 3: Small Matching ⇒ Small Modulator
If $|M| < k$, the vertices participating in M form a $\overline{K_2}$-free modulator of size $p < 2k$. Then apply Proposition 5 to solve Dual Coloring in $O^*(4^k)$ time.

In **Step 1** we obtain M by running a polynomial time algorithm for finding a maximum matching in the complement graph $\overline{G}$. In **Step 2** we observe that if $|M| \geq k$, then we can assign a distinct color for each $\overline{K_2}$ in M to color its two vertices, and then color all remaining vertices in G with different colors to obtain a proper coloring using at most $k + (n-2k) = n-k$ colors overall. In **Step 3** we observe that if $|M| < k$, by maximality deleting each $\overline{K_2}$ in M results in a $\overline{K_2}$-free graph, so Proposition 5 solves the problem.

This strategy of extracting a matching and then performing casework on its size to either identify a solution or enforce more structure is similar to previous work on kernelization for Dual Coloring [7, 17] and applications of crown decomposition [10, Chapter 4].

To solve Dual Coloring faster in $O^*(2^{3k/2})$ time, we work with a structure more complicated than a matching: a triangle packing. The core technical lemma powering our new algorithms is the following result.

▶ **Lemma 6.** *Given a $\overline{K_3}$-free modulator of size p and any positive integer k, the k-Coloring problem can be solved in randomized $O^*(2^p)$ time.*

Since any $\overline{K_3}$-free modulator is also a $\overline{K_2}$-free modulator by definition, Lemma 6 is a stronger version of Proposition 5. We remark that an extension to $\overline{K_4}$-free modularator is unlikely because Coloring is NP-hard on $\overline{K_4}$-free graphs [16].

Step 1: Identify Triangle Packing
Extract a maximal $\overline{K_3}$-packing $\mathcal{T}$ in G.
Step 2: Large Packing ⇒ Easy Instance
If $|\mathcal{T}| \geq k/2$, then report that a proper coloring using at most $(n-k)$ colors exists.
Step 3: Small Packing ⇒ Small Modulator
If $|\mathcal{T}| < k/2$, the vertices participating in $\mathcal{T}$ form a $\overline{K_3}$-free modulator of size $p < 3k/2$. Then apply Lemma 6 to solve Dual Coloring in $O^*(2^{3k/2})$ time.

In **Step 1** we obtain $\mathcal{T}$ in polynomial time by running a greedy algorithm. In **Step 2** we observe that if $|\mathcal{T}| \geq k/2$, then we can assign each $\overline{K_3}$ in $\mathcal{T}$ a different color to color its three vertices, and then color all remaining vertices in G with different colors to obtain a proper coloring using at most $k/2 + (n - 3k/2) = n-k$ colors overall. In **Step 3** we observe that if $|\mathcal{T}| < 3k/2$, by maximality deleting each $\overline{K_3}$ in $\mathcal{T}$ results in a $\overline{K_3}$-free graph, so Lemma 6 solves the problem.

The proof of Proposition 5 from [12] is established using algebraic techniques. The main idea is that given a $\overline{K_2}$-free modulator S of size p, the induced subgraph on $V \setminus S$ is a clique, so every vertex outside of S must be assigned a different color. Solving the k-Coloring problem can then be reduced to a problem similar to bipartite matching, where one must determine for each vertex outside of S the subset of vertices in S it shares a color with (intuitively, the subset of S it is "matched to"). The authors use determinants to enumerate bipartite matchings in an auxiliary graph capturing this question, and apply algorithms for *polynomial sieving* and *fast subset convolution* to find the best matching (corresponding to an optimal coloring) and solve this problem in $O^*(2^p)$ time, comparable to the number of subsets of S that must be considered.

Our proof of Lemma 6 uses similar technical ingredients. Here, because we are given a $\overline{K_3}$-free modulator S instead of a $\overline{K_2}$-free modulator, we can no longer assume that the vertices in $\overline{S} = V \setminus S$ all get different colors. However, we can assume that in a proper coloring of the graph, any given color appears at most twice across the nodes in $\overline{S}$. This then lets us reduce the k-Coloring problem in this context to a problem related to perfect matchings, albeit in a nonbipartite graph. At a high level, we do this by considering all partitions of $\overline{S}$ into sets of size at most two, where these parts represent distinct color classes in $\overline{S}$, and then consider all ways of matching these parts to different subsets of S, corresponding to extending these color classes from $\overline{S}$ to the full graph.

Since we work with matchings in general undirected graphs, we use Pfaffians instead of determinants in our arguments. We also used a Pfaffian-based algebraic approach in recent work on Edge Coloring [1]. On top of this framework, we use fast subset convolution to ensure that we achieve the same $O^*(2^p)$ runtime as before.

Our algorithm for $(\omega + \overline{\mu} - k)$-Coloring follows by employing a similar strategy to the procedure for Dual Coloring outlined above, with a more refined analysis with respect to the relevant structural parameters.

Comparison with the $(n-k)$-Set Cover Result. Alferov et al. [2] achieved an $O^*(2^{3k/2}|\mathcal{F}|)$ algorithm for $(n-k)$-Set Cover on input families $\mathcal{F}$ via a different toolkit. Their procedure likewise begins by extracting a maximal triangle packing and immediately reports a yes-instance when that packing is large (**Step 2** in our outline). When the packing is small, they study the matching structure in the remaining part via the Gallai-Edmonds decomposition. A sufficiently large matching again certifies a solution, and they carefully use this fact to obtain a runtime bound of $O^*(2^{3k/2}|\mathcal{F}|)$. These arguments are tailored to $(n-k)$-Set Cover, and do not lead to an efficient algorithm parameterized by $\overline{K_3}$-free modulator size as in Lemma 6, which is central to proving Theorem 3. For the special case of $(n-k)$-Coloring, where the set $\mathcal{F}$ can be succinctly described as the collection of independent sets in a given graph, the dynamic program employed in [2] still suffers from this issue. Due to this limitation, this previous approach does not directly prove Theorem 1 either.

Organization. In Section 2, we review notation and basic facts about graphs, matrices, and polynomials that will be useful to us. In Section 3 we prove Lemma 6. In Section 4, we apply Lemma 6 to design our algorithm for Dual Coloring and $(\omega + \overline{\mu} - k)$-Coloring (our harder structural parameter variant of Dual Coloring) and prove Theorem 3, and also establish a lower bound for a related problem by proving Theorem 4. We conclude in Section 5 by summarizing our work and mentioning relevant open problems.

2 Preliminaries

General Notation. Given a positive integer a, we let $[a] = \{1, \ldots, a\}$ denote the set of the first a consecutive positive integers.

Graph Notation. We let G denote the input graph on n vertices and m edges. We let V denote the vertex set of G. Given a subset of nodes $S \subseteq V$, we let $G[S]$ denote the induced subgraph of G on S. We let $\mu(G), \omega(G)$, and $\alpha(G)$ denote the size of a maximum matching, clique, and independent set in G respectively. We let K_3 denote the complete graph on three nodes, which we refer to as a *triangle*.

From Coloring to Clique Covers. Given a graph G, we say a *clique cover* of G is a collection $\mathcal{C}$ of vertex-disjoint cliques in G such that every vertex appears in some clique of $\mathcal{C}$. The *size* of this cover is simply $|\mathcal{C}|$, the number of cliques used in the clique cover.

In the p-Clique Cover problem, we are given a graph G on n vertices and are tasked with determining if G has a clique cover of size at most p. For convenience, we refer to $(n-k)$-Clique Cover as the Dual Clique Cover problem. By complementing the graph it is easy to see that p-Coloring and p-Clique Cover are equivalent computational tasks.

▶ **Observation 7** (Clique Cover). *For any integer $p \geq 1$, solving p-Coloring on a graph G is equivalent to solving p-Clique Cover on the complement graph $\overline{G}$.*

Proof. Suppose we have a proper coloring of G. Then the set of vertices assigned any fixed color form an independent set in G, which means they form a clique in $\overline{G}$. Thus the coloring induces a clique cover of the same size. The reverse direction, that a solution to p-Clique Cover on $\overline{G}$ implies a solution to p-Coloring on G, follows by symmetric reasoning. ◀

For convenience, in our proofs we will use Observation 7 to frame our algorithms as solving problems related to Clique Cover instead of Coloring.

Algebraic Preliminaries. Throughout we work over a finite field $\mathbb{F}$ of size $\text{poly}(n)$. Arithmetic operations over $\mathbb{F}$ take $\text{poly}(\log n)$ time.

Given a set S, let $\Pi(S)$ denote the set of perfect matchings on S (i.e., the collection of partitions of S into sets of size exactly two). Given a skew-symmetric matrix $\mathbf{A}$ (i.e., a matrix that equals $\mathbf{A} = -\mathbf{A}^\top$ the negative of its transpose) with rows and columns indexed by a set S, its Pfaffian is defined to be

$$\text{Pf}\,\mathbf{A} = \sum_{M \in \Pi(S)} \text{sgn}(M) \prod_{\{u,v\} \in M} \mathbf{A}[u,v] \tag{1}$$

for a function $\text{sgn}\colon \Pi(S) \to \{-1, 1\}$ whose definition is not relevant here [20, Section 7.3.2].

An arithmetic circuit is a way of constructing a polynomial by starting with its input variables, and iteratively building up more complicated expressions by using the standard operations of addition, multiplication, and division. The size of an arithmetic circuit is the total number of non-scalar operations used to construct its final output polynomial in this way. Our algorithms use the well known fact that Pfaffians admit polynomial-size arithmetic circuits that only use addition and multiplication operations [22].

▶ **Proposition 8** (Pfaffian Construction). *In $\text{poly}(n)$ time we can construct a division-free arithmetic circuit of $\text{poly}(n)$ size over $\mathbb{F}$ whose inputs are indeterminate entries of an $n \times n$ skew-symmetric matrix $\mathbf{A}$, and whose output is the polynomial $\text{Pf}\,\mathbf{A}$.*

We make use of the following classic result, proven for example in [19, Theorem 7.2].

▶ **Proposition 9** (Identity Testing). *Let P be a nonzero polynomial over a finite field $\mathbb{F}$ of degree at most d. If each variable of P is assigned an independent, uniform random value from $\mathbb{F}$, then the corresponding evaluation of P is nonzero with probability at least $1 - d/|\mathbb{F}|$.*

Subset Convolution. Let S be a set of size p. Let α and β be functions from the collection of subsets of S to $\mathbb{F}$.

The *subset convolution* $(\alpha * \beta)$ of α and β is the function defined by setting

$$(\alpha * \beta)(T) = \sum_{\substack{A,B \subseteq S \\ T = A \sqcup B}} \alpha(A)\beta(B) \quad \text{for all } T \subseteq S.$$

For each $v \in S$, introduce a variable y_v. Let $Y = \{y_v\}_{v \in S}$ be this set of variables. Consider polynomials

$$P = \sum_{A \subseteq S} \alpha(A) \prod_{a \in A} y_a \quad \text{and} \quad Q = \sum_{B \subseteq S} \beta(B) \prod_{b \in B} y_b$$

over the quotient ring $R = \mathbb{F}[Y]/\langle (y^2)_{y \in Y} \rangle$. That is, R is the usual ring of polynomials over the variables in Y with coefficients from $\mathbb{F}$, except that whenever we multiply the same variable with itself the resulting product vanishes.

We refer to R as the *squarefree ring* over Y. By definition of polynomial multiplication and the fact that only squarefree monomials survive in R, we get that

$$P \cdot Q = \sum_{T \subseteq S} (\alpha * \beta)(T) \prod_{t \in T} y_t$$

over R. This shows that computing subset convolutions is equivalent to computing products of polynomials over R. The following result then follows from known $O^*(2^p)$ time algorithms for subset convolution [4].

▶ **Proposition 10** (Fast Subset Convolution). *We can compute addition and multiplication over the squarefree ring R on p variables in $O^*(2^p)$ time.*

3 Triangle Modulator

In this section, we prove the following key lemma.

▶ **Lemma 6.** *Given a $\overline{K_3}$-free modulator of size p and any positive integer k, the k-Coloring problem can be solved in randomized $O^*(2^p)$ time.*

Proof. Replace the input graph with its complement. Then by Observation 7, it suffices to show that given a K_3-free modulator S of size p for G and a positive integer k, we can solve k-Clique Cover on G in $O^*(2^p)$ time.

We write $\overline{S} = V \setminus S$. By definition, $G[\overline{S}]$ contains no triangle (i.e., a copy of K_3). This means that for every clique C in G, we have $|C \cap \overline{S}| \in \{0, 1, 2\}$. Given a clique cover $\mathcal{C}$ of G, we say it has type $\vec{t} = (t_0, t_1)$ if for each $i \in \{0, 1\}$ the cover $\mathcal{C}$ contains exactly t_i cliques C satisfying $|C \cap \overline{S}| = i$. Note that if a cover has type $\vec{t}$, then because every vertex appears in a unique clique in the cover, exactly $t_2 = (|\overline{S}| - t_1)/2$ cliques in the cover intersect $\overline{S}$ at two nodes. Consequently, the size of a clique cover with type $\vec{t}$ is exactly

$$t_0 + t_1 + t_2 = t_0 + (t_1 + |\overline{S}|)/2 = t_0 + (t_1 + n - p)/2.$$

We say a type $\vec{t} = (t_0, t_1)$ is *valid* if and only if we have $t_0 + (t_1 + n - p)/2 \leq k$ so that a clique cover of type $\vec{t}$ has size at most k.

Our task is to determine if G contains a clique cover of size at most k. Such a cover has at most $O(k^2) \leq \text{poly}(n)$ valid types $\vec{t}$, and we try out each possible such type and look for a clique cover with exactly that type.

To that end, fix a valid type $\vec{t} = (t_0, t_1)$. We now construct an auxiliary graph H, such that perfect matchings in H encode how clique covers of G with type $\vec{t}$ may restrict to clique covers of $G[\overline{S}]$. We include the vertex set $\overline{S}$ in H, and add all edges from $G[\overline{S}]$ to H. We then introduce a set U of t_1 new vertices, and add edges from every node in U to every vertex in $\overline{S}$. Intuitively, when we take a perfect matching M of H, it will correspond to a clique cover $\mathcal{C}$ of G such that

- for each edge $\{u, v\}$ with $u \in U$, there is a clique $C \in \mathcal{C}$ with $C \cap \overline{S} = \{v\}$, and
- for each edge $\{v, w\} \in M$ with $v, w \in \overline{S}$, there is a clique $C \in \mathcal{C}$ with $C \cap \overline{S} = \{v, w\}$.

With this correspondence in mind, we move to constructing a polynomial that will enumerate clique covers of G.

Let $E(H)$ denote the edge set of H. For each $e \in E(H)$, introduce an indeterminate variable x_e. Now consider the matrix $\mathbf{A}$ whose rows and columns are indexed by nodes of H, with entries defined by

$$\mathbf{A}[v, w] = \begin{cases} x_{vw} & \text{if } \{v, w\} \in E(H) \text{ and } v \prec w \\ -x_{vw} & \text{if } \{v, w\} \in E(H) \text{ and } w \prec v \\ 0 & \text{otherwise} \end{cases}$$

where $(\prec)$ is some fixed ordering on the nodes of H.

By Equation (1) and the definition of $\mathbf{A}$ above, we have

$$\text{Pf}\, \mathbf{A} = \sum_{M \in \Pi(H)} \text{sgn}(M) \prod_{e \in M} x_e \tag{2}$$

where here $\Pi(H)$ denotes the set of perfect matchings in H, and $\text{sgn}(M) \in \{-1, 1\}$ for each choice of M.

Next, we will introduce additional, larger polynomial expressions that we will substitute in for the x_e variables, with the end goal of transforming the matching polynomial $\text{Pf}\, \mathbf{A}$ into a polynomial that enumerates collections of cliques in G.

Go through all subsets $T \subseteq S$, and for each check if T is a clique in G. This lets us determine the collection $\mathcal{F}$ of all cliques in S in $O^*(2^p)$ time overall.

Then for each edge $e \in E(H)$ and clique $C \subseteq S$, we introduce a variable z_{eC}. For each $i \in [t_0]$ and clique $C \subseteq S$, also introduce a variable z_{iC}. Let Z be the set of all these z_{eC} and z_{iC} variables.

Additionally, for each each node $v \in S$, we introduce a variable y_v. Let $Y = \{y_v\}_{v \in S}$ denote this set of k variables.

Now for each $e \in E(H)$ and subset $T \subseteq S$, define the polynomial

$$\Phi_e[T] = \sum_{\substack{C \in \mathcal{F} \\ C \subseteq T}} z_{eC} \prod_{v \in C} y_v. \tag{3}$$

Intuitively this polynomial enumerates those cliques $C \subseteq T$ that can be extended using vertices in $e \cap \overline{S}$ to obtain larger cliques in G.

Also for each $i \in [t_0]$, define the polynomial

$$\Phi_i = \sum_{C \in \mathcal{F}} z_{iC} \prod_{v \in C} y_v. \tag{4}$$

Intuitively these polynomials enumerate cliques $C \subseteq S$ that we do not plan on extending with nodes in $\overline{S}$.

For each vertex $v \in V$, let $N(v)$ denote the set of vertices in S adjacent to v in G. We construct a new matrix $\mathbf{B}$, by starting with the matrix $\mathbf{A}$ and

- for each $e = \{u, v\} \in E(H)$ with $u \in U$ and $v \in \overline{S}$, substituting x_{uv} with $\Phi_e[N(v)]$, and
- for each $e = \{v, w\} \in E(H)$ with $v, w \in \overline{S}$, substituting x_{vw} with $\Phi_e[N(v) \cap N(w)]$.

Now define the polynomial

$$F = (\operatorname{Pf} \mathbf{B}) \cdot \prod_{i=1}^{t_0} \Phi_i \tag{5}$$

over $R[Z]$, where $R = \mathbb{F}[Y]/\left\langle (y^2)_{y \in Y} \right\rangle$ is the squarefree ring defined in Section 2.

▷ Claim 11 (Polynomial Characterization). The polynomial F has a monomial divisible by $\prod_{v \in S} y_v$ if and only if G has a clique cover of type $\vec{t} = (t_0, t_1)$.

Proof. Suppose first that F has a monomial divisible by $\prod_{v \in S} y_v$. Since we work over the squarefree ring R, this monomial is of the form

$$f(Z) \prod_{v \in S} y_v \tag{6}$$

for some polynomial $f \in \mathbb{F}[Z]$. Moreover, from Equation (5) such a monomial in F must have been generated by taking the product of monomials from $\operatorname{Pf} \mathbf{B}$ and each of the Φ_i such that the appearance of variables from Y in these monomials partition Y.

For each $i \in [t_0]$, let C_i be the set of vertices $v \in S$ such that the y_v variable was used by the monomial from Φ_i selected to help generate the expression from Equation (6) in the expansion of the product from Equation (5). Then by the partition property, the C_i are pairwise disjoint. From the definition of Φ_i in Equation (4), we get that each C_i is a clique.

Now, consider the monomial selected from $\operatorname{Pf} \mathbf{B}$ to help generate the expression from Equation (6) in the expansion of the product from Equation (5). By Equation (2) and the definition of $\mathbf{B}$ in terms of $\mathbf{A}$, this monomial corresponds to a matching $M \in \Pi(H)$. Split $M = M_1 \sqcup M_2$ by letting M_b retain the edges in M which have exactly b endpoints in $\overline{S}$ for each $b \in [2]$. Since H has vertex set $\overline{S} \sqcup U$ and $|U| = t_1$, we get that M_1 has exactly t_1 edges, and each of its edges has one endpoint in U and one endpoint in $\overline{S}$. Let $v_1, \ldots, v_{t_1}$ be the nodes in $\overline{S}$ appearing as endpoints of edges in M_1. For each $j \in [t_1]$, let e_j be the edge containing v_j in M_1.

Since in $\mathbf{B}$ we take $\mathbf{A}$ and replace each x_{e_j} variable with $\Phi_{e_j}[N(v_j)]$, the product over the edges of M_1 from Equation (2) contributes the variables y_v precisely for those vertices v appearing in the union of some choice of cliques $D_1, \ldots, D_{t_1} \subseteq S$ with the property that the set $D_j \sqcup \{v_j\}$ is a clique in G for each j. Since we work over R, the cliques C_i and D_j are mutually disjoint. Moreover, since $C_i \subseteq \overline{S}$ for all i, we actually have that the cliques C_i and $D_j \sqcup \{v_j\}$ are collectively vertex-disjoint.

Set $t_2 = (|\overline{S}| - t_1)/2$, and let $\tilde{e}_1, \ldots, \tilde{e}_{t_2}$ be the edges of M_2. For each $\ell \in [t_2]$, let $\tilde{N}_\ell$ be the set of common neighbors in S of the endpoints of the edge $\tilde{e}_\ell$. Since in $\mathbf{B}$ we take $\mathbf{A}$ and replace each $x_{\tilde{e}_\ell}$ variable with $\Phi_{\tilde{e}_\ell}[\tilde{N}_\ell]$, the product over the edges of M_2 from Equation (2) contributes the variables y_v precisely for those vertices v appearing in the union of some choice of cliques $\tilde{D}_1, \ldots, \tilde{D}_{t_2} \subseteq S$ with the property that the set $\tilde{D}_\ell \sqcup e_\ell$ is a clique in G for each ℓ. Since we work over R, the C_i, D_j, and $\tilde{D}_\ell$ are mutually disjoint. Moreover, since $M = M_1 \sqcup M_2$ is a matching in S, we in fact know that the C_i, $D_j \sqcup \{v_j\}$, and $\tilde{D}_\ell \sqcup \tilde{e}_\ell$ are all vertex-disjoint cliques in G.

Combining this vertex-disjoint condition with the facts that M is a perfect matching on $\overline{S}$ and that the monomial in Equation (6) generated by the products of the monomials of all the cliques we have listed so far is divisible by $\prod_{v\in S} y_v$, we get that the C_i, $D_j \sqcup \{v_j\}$, and $\tilde{D}_\ell \sqcup \tilde{e}_\ell$ cliques taken together form a clique cover of G of type $\vec{t} = (t_0, t_1)$ as claimed.

Conversely, suppose G has a clique cover of type $\vec{t} = (t_0, t_1)$. Set $t_2 = (|\overline{S}| - t_1)/2$.

Let $C_1, \ldots, C_{t_0}$ be the cliques in this cover intersecting $\overline{S}$ at zero nodes, $D_1, \ldots, D_{t_1}$ be the cliques in this cover intersecting $\overline{S}$ at one node, and $\tilde{D}_1, \ldots, \tilde{D}_{t_2}$ be the cliques in this cover intersecting $\overline{S}$ at two nodes. For each $i \in [t_0]$, select the monomial from Φ_i in Equation (4) corresponding to the clique C_i.

For each $j \in [t_1]$, let v_j be the unique vertex from $\overline{S}$ in D_j. For each $\ell \in [t_2]$, let $\tilde{e}_\ell$ be the subset of two vertices in $\overline{S}$ contained in $\tilde{D}_\ell$. Since $\tilde{D}_\ell$ is a clique, $\tilde{e}_\ell$ is an edge. Let $\tilde{N}_\ell$ be the set of common neighbors in $\overline{S}$ of the endpoints of $\tilde{e}_\ell$. The v_j and $\tilde{e}_\ell$ are vertex-disjoint and account for all vertices in $\overline{S}$, because we are assuming we are starting with a clique cover. So we can take a perfect matching M of H which includes all the $\tilde{e}_\ell$ edges, and pairs each v_j with a different node in U (here we use the fact that U has exactly t_1 nodes). Let e_j be the edge containing v_j in M for each $j \in [t_1]$.

Then we can select the monomial of $\operatorname{Pf} \mathbf{B}$ corresponding to choosing the summand for M in Equation (2), choosing the summand for the cliques $D_j \setminus \{v_j\}$ in the sums for $\Phi_{e_j}[N(v_j)]$ given by Equation (3), and choosing the summands for the cliques $\tilde{D}_\ell \setminus \tilde{e}_\ell$ in the sums for $\Phi_{\tilde{e}_\ell}[\tilde{N}_\ell]$ given by Equation (3) again.

Then from the clique cover assumption, the product of all the monomials we selected from the Φ_i and $\operatorname{Pf} \mathbf{B}$ will be divisible by $\prod_{v\in S} y_v$. Moreover, the product of the variables from Z appearing in this monomial uniquely recover the cliques C_i, D_j, and $\tilde{D}_\ell$, as well as the matching M, because the z_{iC} variables record the ordering of the C_i cliques, and the z_{eC} variables annotate the edges e of the matchings corresponding to the $D_j \cap \overline{S}$ and $\tilde{D}_\ell \cap \overline{S}$ sets. Thus the monomial generated in this way cannot be cancelled out by any other term in the expansion of F in Equation (5), which proves the desired result. ◁

Having established Claim 11, it suffices to show that we can test that F has a monomial divisible by $\prod_{v\in S} y_v$ in $O^*(2^p)$ time. To do this, we will take a random evaluation of the variables in Z, and then use fast subset convolution over the variables in Y.

For all $i \in [t_0]$ and cliques $C \subseteq S$, we pick independent, uniform random $\xi_{iC} \in \mathbb{F}$. Independently from those values, for all $e \in E(H)$ and $C \subseteq S$ we also pick independent, uniform random $\xi_{eC} \in \mathbb{F}$. For each $i \in [t_0]$, let

$$\varphi_i = \sum_{C\in\mathcal{F}} \xi_{iC} \prod_{v\in C} y_v. \tag{7}$$

be the result of evaluating the Z variables of Φ_i on the ξ_{iC} values. We can compute each φ_i in $O^*(2^p)$ time because we have already precomputed the collection $\mathcal{F}$ of cliques in S.

Similarly, for each $e \in E(H)$ and $T \subseteq S$, let

$$\varphi_e[T] = \sum_{\substack{C\in\mathcal{F}\\ C\subseteq T}} \xi_{eC} \prod_{v\in C} y_v \tag{8}$$

be the result of evaluating the Z variables of $\Phi_e[T]$ on the ξ_{eC} values. Since we precomputed all the cliques contained in S, for any fixed $e \in E(H)$ and $T \subseteq S$ we can use Equation (8) to compute $\varphi_e[T]$ in $O^*(2^p)$ time. In particular, we can compute $\varphi_e[T]$ for all $e \in E(H)$ and $T \subseteq S$ of the form $T = N(v)$ for some $v \in \overline{S}$ or $T = N(v) \cap N(w)$ for some $v, w \in \overline{S}$ in $O^*(2^p)$ time, because there are $\operatorname{poly}(n)$ choices for e and T in this case.

Having computed all these values, we can substitute the x_e for the relevant values $\varphi_e[T]$ values needed to turn $\mathbf{A}$ into $\mathbf{B}$ (with respect to our random evaluation of the variables in Z). Now use Proposition 8 to obtain a polynomial-size, division-free arithmetic circuit for the Pfaffian, and feed the entries of $\mathbf{B}$ we have just computed into it. Each addition and multiplication operation for this arithmetic circuit is now over the squarefree ring R, which by Proposition 10 takes $O^*(2^p)$ time. Since there are only $\text{poly}(n)$ such operations, we compute this random evaluation of $\operatorname{Pf}\mathbf{B}$ over R in $O^*(2^p)$ time overall. We then compute the product of this with all the φ_i again in $O^*(2^p)$ time by Proposition 10, which by Equation (5) gives us the value of the polynomial F under our random evaluation to its Z variables.

Since F has degree at most $\text{poly}(n)$ in its Z variables, applying Proposition 9 to the coefficient of $\prod_{v\in S} y_v$ in F (viewed as a polynomial in Z), we see that by picking the size of the field $\mathbb{F}$ to be a sufficiently large polynomial in n, with high probability the random evaluation of F we computed has the monomial $\prod_{v\in S} y_v$ if and only if the original polynomial F has a monomial divisible by $\prod_{v\in S} y_v$. So by Claim 11 checking if this random evaluation of F has the monomial $\prod_{v\in S} y_v$ solves the k-Coloring problem in $O^*(2^p)$ time with high probability. This proves the desired result. ◀

4 Below Guarantee Parameterizations

In this section, we present our algorithm for Dual Coloring.

▶ **Theorem 1.** Dual Coloring *can be solved in randomized* $O^*(2^{3k/2})$ *time.*

Proof. We replace the graph G with its complement. Then by Observation 7, it suffices to solve the Dual Clique Cover problem.

We first construct a maximal collection $\mathcal{T}$ of vertex-disjoint triangles in G. Since we can find a triangle in G if one exists in polynomial time, we can construct $\mathcal{T}$ by repeatedly finding a triangle T in G, including T in $\mathcal{T}$, then deleting the vertices of T from G and repeating this process, until we determine that no triangles are left. We find at most $n/3$ triangles in this way, so this takes polynomial time overall.

Let $t = |\mathcal{T}|$. Suppose $t \geq k/2$. Then taking the triangles in $\mathcal{T}$ together with the single-node sets consisting of each vertex v not used by a triangle in $\mathcal{T}$ yields a clique cover for G of size $t + (n - 3t) = n - 2t \leq n - k$. Thus, we can solve Dual Clique Cover by returning the cover described above.

Otherwise, we have $t < k/2$. In this case, maximality of $\mathcal{T}$ implies that the set $S = \bigsqcup_{T\in\mathcal{T}} T$ is a K_3-free modulator for G of size $3t$. Then by Observation 7 and Lemma 6 we can solve Dual Coloring in this case in $O^*(2^{3t}) \leq O^*(2^{3k/2})$ time, as desired. ◀

Next, we establish (randomized) fixed-parameter tractability for a smaller parameter.

▶ **Theorem 3** (Parameterization from Stronger Guarantee)**.** *Given an integer* $k \geq 1$*, and a graph* G *that has maximum clique size* ω *and whose complement has maximum matching size* $\overline{\mu}$*, we can solve* $(\omega + \overline{\mu} - k)$-Coloring *in randomized* $O^*(2^{6k})$ *time.*

Proof. We replace the graph G with its complement. Let $\mu = \mu(G)$ and $\alpha = \alpha(G)$ be the size of a maximum matching and maximum independent set in the new graph respectively. Since complementing the graph turns cliques into independent sets, by Observation 7 it suffices to show that we can solve $(\alpha + \mu - k)$-Clique Cover in $O^*(2^{6k})$ time.

We first construct a maximal collection $\mathcal{T}$ of vertex-disjoint triangles in G. Since finding a triangle in G, if it exists, takes polynomial time, and $\mathcal{T}$ can have at most $n/3$ triangles, we can obtain $\mathcal{T}$ in polynomial time using a greedy algorithm that repeatedly finds triangles, includes them in $\mathcal{T}$, deletes the obtained triangles from G, and continues in this fashion until we are left with a triangle-free graph.

Let $t = |\mathcal{T}|$. Suppose that $t \leq 2k$. In this case, define the set $S = \bigsqcup_{T \in \mathcal{T}} T$ of all vertices participating in the triangles of $\mathcal{T}$. Then by maximality of $\mathcal{T}$, the set S is a K_3-free modulator of G of size $3t$. In this case, by Observation 7 and Lemma 6 we can solve $(\alpha + \mu - k)$-Clique Cover in $O^*(2^{3t}) \leq O^*(2^{6k})$ time as claimed.

Otherwise, $t > 2k$. In this case, take $\tilde{\mathcal{T}} \subseteq \mathcal{T}$ consisting of exactly $2k$ triangles, and define $S = \bigsqcup_{T \in \tilde{\mathcal{T}}} T$ to be a set of $6k$ vertices making up $2k$ of the triangles in $\mathcal{T}$. We show that in this case, a clique cover of size at most $(\alpha + \mu - k)$ always exists in G.

Write $\overline{S} = V \setminus S$. Let M_{out} be a maximum matching in $G[\overline{S}]$. We can find M_{out} in polynomial time. Let $\mu_{\text{out}} = |M_{\text{out}}|$ be the size of this matching. Let I be the set of vertices in $\overline{S}$ that do not appear as an endpoint of an edge in M_{out}. By maximality of M_{out}, the set I forms an independent set in G. Write $\alpha_{\text{out}} = |I|$.

▷ Claim 12. We have $\mu_{\text{out}} + \alpha_{\text{out}} \leq \alpha + \mu - 3k$.

Proof. We will prove the claim by establishing several inequalities concerning structural parameters in G.

First, construct a maximum matching M_{in} in $G[S \sqcup I]$ in polynomial time. Let $\mu_{\text{in}} = |M_{\text{in}}|$ be the size of this matching. Since M_{out} is in $G[\overline{S} \setminus I]$ by definition, we see that $M_{\text{in}} \sqcup M_{\text{out}}$ is a matching in G of size $(\mu_{\text{in}} + \mu_{\text{out}})$.

Since μ is the size of a maximum matching in G, we get that

$$\mu_{\text{in}} + \mu_{\text{out}} \leq \mu. \tag{9}$$

Note that since α is the maximum size of independent sets in G, we have

$$\alpha_{\text{out}} \leq \alpha. \tag{10}$$

Now, by maximality of M_{in} in $G[S \sqcup I]$, we know that the set of vertices in $S \sqcup I$ not participating as an endpoint in M_{in} forms an independent set in G. This set then has size $(\alpha_{\text{out}} + 6k - 2\mu_{\text{in}})$. Since α is the maximum size of independent sets in G, we have

$$\alpha_{\text{out}} + 6k - 2\mu_{\text{in}} < \alpha. \tag{11}$$

Now, we can write

$$\mu_{\text{out}} + \alpha_{\text{out}} = \mu_{\text{out}} + (1/2)\alpha_{\text{out}} + (1/2)\alpha_{\text{out}}. \tag{12}$$

By applying Equations (9)–(11) to the first through third terms on the right-hand side of Equation (12) respectively, we can bound

$$\mu_{\text{out}} + \alpha_{\text{out}} \leq (\mu - \mu_{\text{in}}) + (1/2)\alpha + (1/2)(\alpha + 2\mu_{\text{in}} - 6k) = \alpha + \mu - 3k$$

which proves the desired result. ◁

By definition, every vertex in $\overline{S}$ either appears as an endpoint of M_{out}, or belongs to I. Then using the triangles in $\tilde{\mathcal{T}}$ to cover the vertices in S, and taking the edges from M_{out} and the individual nodes from I to cover the vertices in $\overline{S}$, we obtain a clique cover for G of size

$$2k + \mu_{\text{out}} + \alpha_{\text{out}} \leq \alpha + \mu - k$$

by Claim 12. So we can solve $(\alpha + \mu - k)$-Clique Cover just by returning this cover. ◀

Curiously, previous work showed that Induced Matching (the problem of finding a 1-regular induced subgraph on k vertices) is also fixed-parameter tractable in k for the same below-guarantee parameter $\alpha + \mu - k$ as in the proof above [15].

Hardness. We now present our lower bound, demonstrating that although Theorem 3 shows we can get an efficient parameterized algorithm for graph coloring below $(\omega + \overline{\mu})$, we do not expect an analogous algorithm to exist for coloring below the quantity $\overline{\mu}$ on its own.

▶ **Theorem 4.** *The $(n/2 - k)$-*Coloring *problem is* W[1]*-hard in the parameter k on graphs with n vertices whose complement has a perfect matching.*

Proof. By replacing the graph with its complement and applying Observation 7, it suffices to show that $(n/2 - k)$-Clique Cover is W[1]-hard on graphs containing a perfect matching.

We prove this result by reducing from k-Colored Clique. In this problem, we are given a k-partite graph G on kn vertices, with vertex set $V = V_1 \sqcup \cdots \sqcup V_k$ partitioned into parts V_i consisting of n nodes each, and are tasked with determining if G contains a clique on k vertices. This problem is known to be W[1]-hard [9, Theorem 13.25].

Take an instance G of k-Colored Clique.

For each $i \in [k]$, we order the vertices in $V_i = \{v_{i1}, \ldots, v_{in}\}$.

We construct a larger graph $\tilde{G}$ that contains G as a subgraph, in addition to some auxiliary nodes and edges we describe next. For each $i \in [k+2]$, we introduce a new node u_i in $\tilde{G}$. For each $i \in [k]$ and $j \in [n-1]$, we introduce a new node w_{ij} in $\tilde{G}$. We add edges between all the u_i nodes. For all $i \in [k]$, we add an edge from u_i to v_{i1}. For all $i \in [k]$ and $j \in [n-1]$ we add edges from w_{ij} to v_{ij} and $v_{i(j+1)}$.

Conceptually, this construction of $\tilde{G}$ from G involves adding a new clique of size $(k+2)$ to the graph on the u_i vertices, and for each $i \in [k]$ introducing a path

$$P_i = \langle u_i, v_{i1}, w_{i1}, v_{i2}, w_{i2}, \ldots, w_{i(n-1)}, v_{in} \rangle \tag{13}$$

beginning at u_i, and alternating between nodes in V_i and W_i according to the orders of the vertices in these sets. By design, for any $j \in [n]$, we can always split the path

$$P_i = \langle u_i \rangle \diamond A_{ij} \diamond \langle v_{ij} \rangle \diamond B_{ij} \tag{14}$$

into its first node, a path A_{ij} on an even number of vertices, the node v_{ij}, and a suffix B_{ij} on an even number of vertices. Moreover, A_{ij} and B_{ij} consist of $2n-2$ nodes altogether in this decomposition.

Note that the constructed graph $\tilde{G}$ has a perfect matching, by taking the edges $\{u_i, v_{i1}\}$ for all $i \in [k]$ and $\{w_{ij}, v_{i(j+1)}\}$ for all $i \in [k]$ and $j \in [n-1]$.

From its definition, $\tilde{G}$ has $N = 2kn + 2$ nodes. Set the parameter $\ell = k(n-1) + 2$.

▷ Claim 13. G has a clique on k vertices if and only if $\tilde{G}$ has a clique cover of size ℓ.

Proof. Suppose that G has a clique of size k. Let $(v_{1j_1}, \ldots, v_{nj_n}) \in V_1 \times \cdots \times V_k$ be the k-tuple of vertices participating in this clique.

For each $i \in [k]$, by Equation (14) we can split each path P_i into

$$\langle u_i \rangle \diamond A_{ij_i} \diamond \langle v_{ij_1} \rangle \diamond B_{ij_1}$$

its first node, two paths A_{ij_i} and B_{ij_1} on even numbers of vertices with $2n-2$ nodes total, and the node v_{ij_1}. For each $i \in [k]$, we can cover the nodes in A_{ij_i} and B_{ij_i} using the endpoints of the edges from a matching of size $(n-1)$. This shows that with $k(n-1)$ cliques, we can cover all vertices in $\tilde{G}$ in a disjoint fashion, except the u_i and v_{ij_i} nodes. We include all the u_i nodes in one clique and all the v_{ij_i} nodes in another clique to then obtain a clique cover of size $\ell = k(n-1) + 2$ of $\tilde{G}$ as claimed.

Conversely, suppose $\tilde{G}$ has a clique cover of size ℓ. Note that the w_{ij} vertices, with indices ranging over $i \in [k]$ and $j \in [n-1]$, form an independent set. Consequently, covering the w_{ij} requires using $k(n-1)$ distinct cliques. Now, each w_{ij} has degree two in $\tilde{G}$, and the neighbors of w_{ij} are non-adjacent in $\tilde{G}$ by definition. Consequently, the cliques used to cover the w_{ij} collectively cover at most $2k(n-1)$ vertices in $\tilde{G}$.

In the assumed clique cover of size ℓ from $\tilde{G}$, this implies that a set of $\ell - k(n-1) = 2$ distinct cliques are used to cover the at least $N - 2k(n-1) = 2k+2$ remaining nodes in $\tilde{G}$ not covered by the cliques that contain the w_{ij} vertices. Since node u_{k+2} is adjacent to only other u_i nodes, the clique that covers u_{k+2} must solely consist of u_i nodes. Without loss of generality, we may then assume the clique covering u_{k+2} contains all u_i nodes. After removing the nodes covered by this clique, we have one clique left, that must be used to cover the at least $(2k+2) - (k+2) = k$ remaining nodes in $\tilde{G}$. Moreover, this clique uses no u_i nodes or w_{ij} nodes. Consequently, the last clique must be a clique of size k in the v_{ij} nodes, which pulls back to the a clique of size k in G. This proves the claim. ◁

Observe that $N/2 - \ell = (k+1)$, so that $\ell = N/2 - (k+1)$.

By Claim 13, solving k-Colored Clique problem on graphs with n vertices reduces to solving $(N/2-(k+1))$-Clique Cover on graphs with $N = 2kn+2$ nodes and a perfect matching. Hence the reduction is efficient enough to imply that $(n/2-k)$-Clique Cover is W[1]-hard in graphs with perfect matchings, which proves the desired result. ◀

5 Conclusion

In this paper, we presented a faster parameterized algorithm for Dual Coloring, improving the runtime from $O^*(4^k)$ to $O^*(2^{3k/2}) \leq O^*(2.83^k)$. We also introduced a new below-guarantee parameterization for graph coloring, which can be viewed as a harder version of Dual Coloring. We showed that this problem can be solved in $2^{O(k)}\operatorname{poly}(n)$ time as well, and noted that a closely related parameterization is in contrast W[1]-hard.

The most relevant open question to this work is: what is the true parameterized complexity of Dual Coloring? The current best algorithms for k-Coloring take $O^*(2^n)$ time for general k, and without improving this runtime we cannot hope to solve Dual Coloring in faster than $O^*(2^k)$ time. Can we indeed achieve a $O^*(2^k)$ runtime for Dual Coloring?

A natural strategy to obtain faster algorithms for Dual Coloring would be to try and strengthen Lemma 6 further, by obtaining efficient coloring algorithms parameterized by H-free modulators for larger pattern graphs H. For example, obtaining an analogue of Lemma 6 for $\overline{K_4}$-free modulators would immediately imply a faster Dual Coloring algorithm by following the framework outlined in Section 1.1. Unfortunately, this simple attempt at generalizing Lemma 6 does not seem possible. This is because solving k-Coloring in graphs without induced copies of $\overline{K}_4$ is NP-hard in general [16, Theorem 1], so that assuming P $\neq$ NP we cannot hope to solve k-Coloring in graphs with $\overline{K}_4$-free modulators of size p in $f(p)\operatorname{poly}(n)$ time, for any function $f(\cdot)$. As mentioned in the Section 1, $(n-k)$-Set Cover can be solved in $O^*(2^{3k/2})$ time using a different method. Can this approach be combined with our triangle packing argument to obtain a faster Dual Coloring algorithm?

Another interesting research direction is to obtain faster algorithms for *multiplicative* below-guarantee graph coloring. For any parameter $\delta \in [1/3, 1]$, by setting $k = (1-\delta)n$ in Theorem 1 we get that (δn)-Coloring can be solved in $O^*(2^{(1-(\delta-1/3)n})$ time. Using very different techniques, previous work showed that for $\delta \in [0,1]$ it is possible to distinguish between the cases where the input graph G admits a proper coloring with $(\delta n - 1)$ colors

and the case where any proper coloring of G needs at least $(\delta n + 1)$ colors in $O^*(2^{(1-\Omega(\delta^4))n})$ time [21, Theorem 1.2]. Can the dependence on δ be improved and the need for additive approximation here be removed, to solve (δn)-Coloring in $O^*(2^{(1-\Omega(\delta))n})$ time in general?

Finally, we showed that for Dual Coloring and its harder variant, constructing a maximal triangle packing instead of a maximal matching can accelerate algorithms for these problems. Can this method be used more generally to help improve parameterized algorithms or kernelization for other problems where the current best methods rely on matching-based techniques such as crown decomposition?

References

1. Shyan Akmal and Tomohiro Koana. Faster edge coloring by partition sieving. In *Proceedings of the 42nd International Symposium on Theoretical Aspects of Computer Science, STACS 2025*, volume 327 of *LIPIcs*, pages 7:1–7:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPICS.STACS.2025.7.
2. Vasily Alferov, Ivan Bliznets, and Kirill Brilliantov. Parameterization of (Partial) Maximum Satisfiability above Matching in a Variable-Clause Graph. In *Proceedings of the 38th AAAI Conference on Artificial Intelligence, AAAI 2024*, pages 7918–7925. AAAI Press, 2024. doi:10.1609/AAAI.V38I8.28628.
3. Kenneth Appel and Wolfgang Haken. Every planar map is four colorable. I. Discharging. *Illinois J. Math.*, 21(3):429–490, 1977. URL: https://projecteuclid.org/euclid.ijm/1256049012.
4. Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets Möbius: fast subset convolution. In *Proceedings of the 39th annual ACM symposium on Theory of computing, STOC 2007*. ACM, June 2007. doi:10.1145/1250790.1250801.
5. Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set Partitioning via Inclusion-Exclusion. *SIAM Journal on Computing*, 39(2):546–563, January 2009. doi:10.1137/070683933.
6. Béla Bollobás, Paul A. Catlin, and Paul Erdös. Hadwiger's Conjecture is True for Almost Every Graph. *Eur. J. Comb.*, 1(3):195–199, 1980. doi:10.1016/S0195-6698(80)80001-1.
7. Benny Chor, Mike Fellows, and David W. Juedes. Linear Kernels in Linear Time, or How to Save k Colors in $O(n^2)$ Steps. In *Proceedings of the 30th International Workshop on Graph-Theoretic Concepts in Computer Science*, volume 3353 of *WG 2004*, pages 257–269. Springer, 2004. doi:10.1007/978-3-540-30559-0_22.
8. Maria Chudnovsky, Neil Robertson, Paul D. Seymour, and Robin Thomas. The strong perfect graph theorem. *Ann. Math.*, 164(1):51–229, 2006. doi:10.4007/annals.2006.164.51.
9. Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer International Publishing, 2015. doi:10.1007/978-3-319-21275-3.
10. Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: Theory of Parameterized Preprocessing*. Cambridge University Press, December 2018. doi:10.1017/9781107415157.
11. M.R. Garey, D.S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, February 1976. doi:10.1016/0304-3975(76)90059-1.
12. Gregory Gutin, Diptapriyo Majumdar, Sebastian Ordyniak, and Magnus Wahlström. Parameterized Pre-Coloring Extension and List Coloring Problems. *SIAM Journal on Discrete Mathematics*, 35(1):575–596, January 2021. doi:10.1137/20m1323369.
13. Gregory Gutin and Matthias Mnich. A Survey on Graph Problems Parameterized Above and Below Guaranteed Values, 2024. arXiv:2207.12278v2.
14. Tommy R. Jensen and Bjarne Toft. *Graph Coloring Problems*. Wiley, 2011.
15. Tomohiro Koana. Induced matching below guarantees: Average paves the way for fixed-parameter tractability. In *Proceedings of the 40th International Symposium on Theoretical Aspects of Computer Science, STACS 2023*, volume 254 of *LIPIcs*, pages 39:1–39:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.STACS.2023.39.

16 Daniel Král, Jan Kratochvíl, Zsolt Tuza, and Gerhard J. Woeginger. Complexity of coloring graphs without forbidden induced subgraphs. In *Proceedings of the 27th International Workshop on Graph-Theoretic Concepts in Computer Science, WG 2001*, volume 2204 of *Lecture Notes in Computer Science*, pages 254–262. Springer, 2001. doi:10.1007/3-540-45477-2_23.

17 Wenjun Li, Yang Ding, Yongjie Yang, and Guozhen Rong. A $(2+\varepsilon)k$-vertex kernel for the dual coloring problem. *Theoretical Computer Science*, 868:6–11, May 2021. doi:10.1016/j.tcs.2021.03.035.

18 László Lovász. Coverings and colorings of hypergraphs. *Proceedings of the 4th Southeastern Conference of Combinatorics, Graph Theory, and Computing*, pages 3–12, 1973. URL: https://cir.nii.ac.jp/crid/1572261549354589440.

19 Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, August 1995. doi:10.1017/cbo9780511814075.

20 Kazuo Murota. *Matrices and matroids for systems analysis*, volume 20. Springer Science & Business Media, 1999.

21 Jesper Nederlof. Finding Large Set Covers Faster via the Representation Method. In *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICS.ESA.2016.69.

22 Günter Rote. *Division-Free Algorithms for the Determinant and the Pfaffian: Algebraic and Combinatorial Approaches*, pages 119–135. Springer Berlin Heidelberg, 2001. doi:10.1007/3-540-45506-x_9.

A Dimension-Reducing Fréchet Simplification Oracle

Boris Aronov
Department of Computer Science and Engineering, Tandon School of Engineering, New York University, Brooklyn, NY, USA

Tsuri Farhana
Department of Computer Science, Ben-Gurion University, Beer Sheva, Israel

Matthew J. Katz
Department of Computer Science, Ben-Gurion University, Beer Sheva, Israel

Indu Ramesh
Department of Computer Science and Engineering, Tandon School of Engineering, New York University, Brooklyn, NY, USA

Abstract

Let P be a polygonal curve with n vertices in the plane. We construct a data structure of size $O(n \log n)$ suited for simplification queries of the following kind. Given a query line ℓ and an integer $k \geq 1$, find a curve Q on ℓ with at most k vertices that minimizes the discrete Fréchet distance to P, among all such curves. Using our data structure, a query can be handled in $O(k^2 \log^3 n + k \log^4 n)$ time.

More generally, a geometric tree T on n vertices in the plane can be preprocessed into a near-linear-size structure so that, given a pair u, v of its vertices, a line ℓ, and an integer $k \geq 1$, one can find a curve Q on ℓ with at most k vertices that minimizes the discrete Fréchet distance to the path from u to v in T, in time $O(k^2 \operatorname{polylog} n)$.

For the general *dimension-reduction* problem, where P is a curve in $\mathbb{R}^d$ ($d \geq 3$), $0 < \varepsilon_0 < 1$ is a real parameter, and a query specifies a g-flat h ($1 \leq g \leq d-1$) and an integer $k \geq 1$, we construct a data structure of size $O(n \log n + f(\varepsilon_0)n)$, where $f(\varepsilon_0) = (1 + 1/\varepsilon_0)^{(d-1)/2}$, that allows us to find a curve Q on h with at most k vertices, whose discrete Fréchet distance to P is at most $1 + \varepsilon_0$ times the distance of Q^* to P, where Q^* is such a curve that minimizes the distance to P. The query handling time is $O(f(\varepsilon_0)k^2 \log^2 n)$.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Computational geometry, discrete Fréchet distance, curve simplification oracle, restricted minimum enclosing disk queries

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.6

Funding *Boris Aronov*: Work partially supported by NSF Grant CCF-20-08551.
Matthew J. Katz: Work partially supported by Grant 2019715/CCF-20-08551 from the US-Israel Binational Science Foundation/US National Science Foundation.
Indu Ramesh: Work supported by NSF Grant CCF-20-08551.

1 Introduction

Curve simplification is an important and extensively studied problem in computational geometry and related fields. Here, we restrict our attention to curve simplification with the quality of the simplification measured using the discrete Fréchet distance between the original curve and its simplification. See below for a comprehensive survey of research in this area.

In this paper, we introduce a novel class of curve simplification problems, in which the simplified curve is required to lie within a specified subspace or region. More formally, let P be a polygonal curve in $\mathbb{R}^d$. We are interested in problems where, in addition to standard

© Boris Aronov, Tsuri Farhana, Matthew J. Katz, and Indu Ramesh;
licensed under Creative Commons License CC-BY 4.0

parameters such as a threshold on the distance between P and the desired simplification Q or a bound on the length (i.e., number of vertices) of Q, Q must also reside in a designated subspace or region. Moreover, since it is often desirable to find the optimal simplification of P for multiple subspaces or regions, we concentrate on the multi-shot formulation of these problems.

These questions can also be viewed as *dimension-reduction* problems. That is, we wish to replace an input d-dimensional curve by a curve in some lower-dimensional space, while minimizing the Fréchet distance to the input curve.

Specifically, we focus on the following problem. Let P be a polygonal curve in $\mathbb{R}^d$. Preprocess P into a compact data structure to efficiently support queries that specify a g-flat[1] h ($1 \leq g \leq d-1$) and an integer $k \geq 1$, and request a curve Q on h of length at most k that minimizes the discrete Fréchet distance to P. For $d = 2$, the g-flat h is simply a line ℓ in the plane, and we present an efficient solution to the problem, as well as to the more general problem in which the input is a geometric tree T rather than a curve P. In the latter problem, a query also specifies the path in the tree to which the query applies. For $d > 2$, we present an efficient solution that provides approximate answers to queries, that is, given h and k, it returns a curve Q on h with at most k-vertices which is approximately the best such curve on h, in terms of its discrete Fréchet distance to P.

Basic definitions. In this paper we view a finite sequence of points $P = (p_1, \ldots, p_n)$ in $\mathbb{R}^d$ as a polygonal curve; we refer to n as the *size* or *length* of P and the points as its *vertices*.

Let $P = (p_1, \ldots, p_n)$ and $Q = (q_1, \ldots, q_m)$ be polygonal curves in $\mathbb{R}^d$. A *legal walk* of P and Q (or just *walk*, for short) is a monotonically increasing sequence of pairs $(r_1, \ldots, r_l)$, where $r_1 = (p_1, q_1)$, $r_l = (p_n, q_m)$, and, in general, when advancing from $r_k = (p_i, q_j)$, r_{k+1} is one of the following: (p_{i+1}, q_j) (provided $i < n$), (p_i, q_{j+1}) (provided $j < m$), or (p_{i+1}, q_{j+1}) (provided $i < n$ and $j < m$). The *cost* of a walk is the maximum distance among the distances between the vertices of each pair $r_k = (p_i, q_j)$ in the walk. The *discrete Fréchet distance* $d_{\mathrm{dF}}(P, Q)$ between P and Q is the cost of the minimum-cost walk of P and Q.

Given a g-flat h, $1 \leq g \leq d-1$, we say that Q is a *k-vertex curve on h* if $Q = (q_1, \ldots, q_k)$ is a sequence of k points on h. An *at-most-k-vertex curve on h* is a k'-vertex curve on h with $1 \leq k' \leq k$.

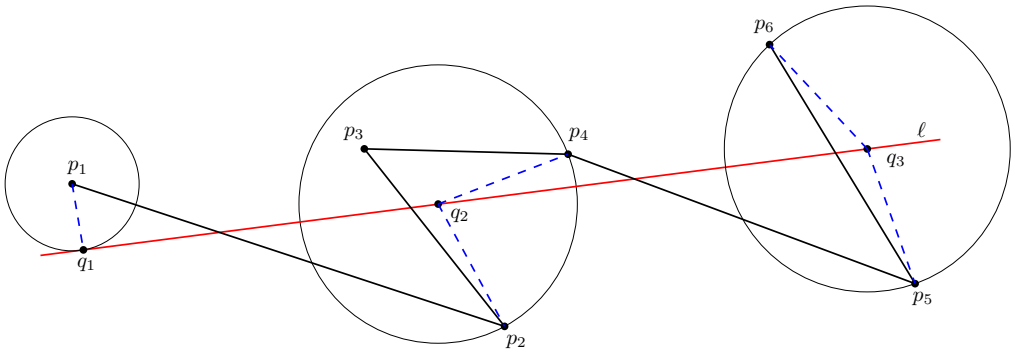


Figure 1 A polygonal curve $P = (p_1, \ldots, p_6)$ and a query line ℓ. In this example, $k = 3$ and $Q = (q_1, q_2, q_3)$ is the returned curve.

[1] That is, a g-dimensional flat – a g-dimensional affine subspace of $\mathbb{R}^d$.

Main problem statement. In this paper we study the following problem. Given a curve P in $\mathbb{R}^d$ as above, preprocess it into a near-linear size data structure to efficiently support queries of the following type: "Given a g-flat h, $1 \leq g \leq d-1$, and integer $k \geq 1$, find an at-most-k-vertex curve Q on h minimizing $d_{\mathrm{dF}}(P, Q)$." We first focus on the planar version, where P is a curve in the plane and queries are lines. See Figure 1 for an example.

We also consider the more general setting, still in the plane, where the input is a geometric tree T rather than a curve P, and the queries are of the form: "Given two nodes p, q of the tree, a line ℓ, and integer k, find an at-most-k-vertex curve Q on ℓ minimizing the discrete Fréchet distance between Q and the path from p to q in T."

In other words, we first study problems in which we need to preprocess a given curve P (or tree T) in the plane, so that one can efficiently handle *simplification* queries with respect to a query line. The maximum length k of the desired simplification is also a parameter of the query.

Next, we further generalize the problem so that a query may specify an arbitrary polygonal region in which Q must reside; query time will of course depend on the complexity of the region.

Finally, we consider the problem in three and higher dimensions, where P is a curve in $\mathbb{R}^d$, $d \geq 3$, and queries are g-flats, $1 \leq g \leq d-1$. Since we are interested in a near-linear size data structure, we resort to approximate queries (see discussion in Section 6), with a prespecified error parameter $\varepsilon_0 > 0$. That is, the answer to a query is an at-most-k-vertex curve Q on h, such that $d_{\mathrm{dF}}(P, Q) \leq (1 + \varepsilon_0) d_{\mathrm{dF}}(P, Q^*)$, where Q^* is such a curve minimizing the distance to P.

Our results. We present efficient solutions to these problems. Specifically, given P or T in the plane, we construct a data structure of size $O(n \log n)$ that allows us to answer a query in $O(k^2 \operatorname{polylog} n)$ time; refer to Theorem 7 in Section 3 and Theorem 8 in Section 4 for the exact statements. As an intermediate result, we present an algorithm for the corresponding decision problem (with respect to a curve P or tree T). For example, given a line ℓ and distance r, we can determine in $O(k \log^2 n)$ time whether there exists an at-most-k-vertex curve on ℓ within discrete Fréchet distance at most r from P, and if so, return such a curve Q with fewest vertices. In Section 5 we extend the data structure so that a query may specify a polygonal region in the plane that restricts the location of the curve Q.

In higher dimensions, given P in $\mathbb{R}^d$ and $0 < \varepsilon_0 < 1$, we construct a data structure of size $O(n \log n + \frac{n}{\varepsilon^{(d-1)/2}})$, where $\varepsilon = \frac{\varepsilon_0}{1+\varepsilon_0}$, that allows us to answer a query approximately in $O(\frac{k^2 \log^2 n}{\varepsilon^{(d-1)/2}})$ time, that is, the discrete Fréchet distance between P and the curve that we return is at most $1 + \varepsilon_0$ times the distance between P and the optimal curve; refer to Theorem 12 in Section 6 for the exact statement. Our solution uses coresets.

Related work on curve simplification. We focus here on (global) simplification under the Fréchet distance in the plane, where the quality of a simplification is the Fréchet distance between the original curve P and its simplification Q.

There are two main versions of the simplification problem. In the first, we are given a threshold δ and the goal is to compute a curve Q of minimum length, such that the distance between P and Q is at most δ. In the second, we are given an integer $k \geq 1$ and the goal is to compute a curve Q of length at most k that minimizes the distance to P. Moreover, each of the two main versions gives rise to four standard variants, depending on whether we are using the continuous or discrete Fréchet distance, and whether Q is *restricted* (i.e., a subsequence of P) or *unrestricted* (i.e., any sequence of points); Van de Kerkhof et al. [18] also consider an intermediate *curve-restricted* option.

The following results are for the first version under the continuous Fréchet distance. Bringmann and Chaudhury [7] presented an algorithm with running time $O(n^3)$ for restricted simplification and established a conditional cubic lower bound. An algorithm with the same running time was also proposed by Van de Kerkhof et al. [18] (see also Van Kreveld et al. [19]). For unrestricted simplification, Guibas et al. [12] provide an algorithm with running time $O(n^2 \log^2 n)$, and Agarwal et al. [2] provide an $O(n \log n)$ algorithm, which returns a simplification Q' of length at most the length of an optimal simplification with threshold $\delta/8$. The latter two statements can be deduced from the corresponding papers, as observed by Van de Kerkhof et al. [18].

Bereg et al. [4] studied both restricted and unrestricted simplification for each of the two main versions, under the discrete Fréchet distance in three-dimensional space. For the first version, they compute an optimal unrestricted simplification in $O(n \log n)$ time and an optimal restricted simplification in $O(n^2)$ time. For the second version, they compute an optimal unrestricted simplification in $O(kn \log n \log(n/k))$ time, where k is the length of the computed simplification, and an optimal restricted simplification in $O(n^3)$ time. Also under the discrete Fréchet distance, Fan et al. [10] considered the so called *chain-pair simplification problem* in the plane, where the goal is to simultaneously compute restricted simplifications of length k for two input curves, such that the distance between the simplifications themselves is also below some threshold.

Finally, under the continuous Fréchet distance, Driemel and Har-Peled [8] presented a near-linear time algorithm that computes a permutation of the vertices of P, such that any prefix of $2k-1$ vertices of this permutation is an approximation (up to a constant factor) of P compared to any polygonal curve with k vertices. Subsequently, Filtser [11] slightly improved the approximation factor under the discrete Fréchet distance.

More related work. A series of papers [16, 15, 5] culminates in an algorithm that preprocesses an n-point set S in the plane in linear space and $O(n \log n)$ time to support $O(\log n)$ time queries of the form "What is the smallest circle centered on the query line and enclosing S?"

There is also a substantial amount of work on so-called *range-aggregate* queries. Given an n-point set S in the plane, we want to preprocess it for queries of the following type: compute some aggregate quantity about the points of S contained in an axis-aligned query rectangle. While range counting, reporting, and more general semigroup queries have been widely studied [1], some types of queries are not readily decomposable (in the sense of the answer being easily synthesized from the answer for smaller queries). For example, Brass et al. [6] discuss computing the area (or perimeter) of the convex hull, width, and smallest enclosing disk of the points of S in the query rectangle. They describe algorithms for preprocessing S into a data structure of near-linear size with polylogarithmic-time queries, for all of the above queries except for the width; the width data structure is more expensive. Gupta et al. [13] explore several variants of closest-pair-in-range problems. For example, they show a near-linear-space structure for computing the closest pair of points within the query rectangle in two dimensions with polylogarithmic query time. Our observation in Remark 4 is a further generalization in this direction: Given a point set in the plane, we can preprocess it so that, given a query rectangle *and* a query line, we can determine the smallest enclosing circle of the points in the rectangle, among all circles centered on the line.

Overview and organization. We first consider the planar version of the dimension-reduction problem. To answer a *decision* query of the form "given a line ℓ, an integer k, and a distance $r > 0$, determine whether there exists an at-most-k-vertex curve on ℓ within Fréchet distance

at most r from P", we need to repeatedly perform the primitive operation $\texttt{prefix}_\ell(P[i,n],r)$ for different values of i: Find the longest (contiguous) subcurve of P, beginning at p_i, that fits into a disk of radius r centered at a point of ℓ. In Section 2, we describe our data structure for the planar version and then use it to implement $\texttt{prefix}_\ell(P[i,n],r)$ in $O(\log^2 n)$ time. Our data structure is obtained by combining known data structures in a non-trivial manner. In Section 3, we obtain our first main result, i.e., the solution for the planar version of the dimension-reduction problem. Using the data structure and primitive operations developed in Section 2, we present efficient algorithms for handling both decision queries and "regular" queries, where the algorithm for regular queries is based on the one for decision queries and on the primitive operation $\text{radius}_\ell(P[i,j])$ (developed in Section 2) which, given a (contiguous) subcurve of P, returns the radius of the smallest disk centered at a point of ℓ that contains the subcurve. In Section 6, we obtain our second main result, i.e., the solution for the d-dimensional version, $d \geq 3$, of the dimension-reduction problem. The algorithms in this general version are similar to those in the planar version, however, we need to replace the exact primitive operations above by their approximate d-dimensional counterparts. These latter operations are implemented using coresets. Finally, in Sections 4 and 5, we extend our result for the planar version in two directions. In Section 4, the input is a geometric tree T (rather than a curve P), and queries specify, in addition to ℓ and k, a path in T, by providing its start and end vertices. In Section 5, queries specify a polygonal region rather than a line ℓ, to contain the simplified curve.

2 Facts and tools

2.1 Facts

For two points p, q in the plane, let $d(p,q)$ be the Euclidean distance between them; many of our observations generalize to other norms, but in this version we focus on the Euclidean metric, for simplicity of presentation.

For a point set Q, let $\text{radius}(Q)$ be the radius of the *minimum enclosing circle* (or *mec*, for short) of Q, and let $\text{radius}(Q,p)$ be the radius of the mec of Q centered at p. Moreover, let $\text{radius}_\ell(Q)$ be the radius of the mec of Q centered at a point of line ℓ, and let $\text{radius}_\ell(Q,x)$ be the radius of the mec of Q centered at $x \in \ell$, as a function of x. We give the proof of the following standard fact below, for completeness.

▶ **Fact 1.** *Both* $\text{radius}(Q,x)$ *and* $\text{radius}_\ell(Q,x)$ *are convex and have a unique minimum.*

Proof. Since $\text{radius}(Q,x) = \max_{q\in Q} d(q,x)$, and $d(q,x)$ is a convex function of x, $\text{radius}(Q,x)$ is convex. Therefore $\text{radius}_\ell(Q,x)$ is also convex, as a restriction of the convex function $\text{radius}(Q,x)$ to a line.

As for uniqueness of minimum, we prove it for $\text{radius}_\ell(Q,x)$; it also proves the claim for $\text{radius}(Q,x)$.

For a contradiction, suppose p and $q \neq p$ are both minima of $\text{radius}_\ell(Q,x)$. Then the disks D_p, D_q of radius $r := \text{radius}_\ell(Q,p) = \text{radius}_\ell(Q,q)$ centered at p and q, respectively, each cover Q. Therefore Q is contained in the lens $D_p \cap D_q$. Let s be a point on the open segment $pq \subset \ell$. Then the disk centered at s and just covering the lens $D_p \cap D_q$ contains Q and has a smaller radius, contradicting the assumption that p and q were minima of $\text{radius}_\ell(Q,x)$. ◀

2.2 Primitives in the plane

We now list the operations that will be useful in solving the two-dimensional problems addressed in this paper. Each of the operations below depends on the query line ℓ.

- $\texttt{disk}_\ell(Q)$: The smallest-radius disk centered at a point of ℓ and containing a set Q of points; let $\text{radius}_\ell(Q)$ and $\text{center}_\ell(Q)$ denote its radius and center, respectively.
- $\texttt{feasible}_\ell(Q, r)$: The interval of ℓ (which may be empty) that is the locus of centers of disks of radius r containing Q.
- $\texttt{fits?}_\ell(Q, r)$: Is there a disk of radius r centered at a point of ℓ and containing a set Q of points? Equivalent to $\texttt{feasible}_\ell(Q, r) \neq \emptyset$ or to $\text{radius}_\ell(Q) \leq r$; the former version is sometimes more efficient.
- $\texttt{prefix}_\ell(S, r)$: For a sequence S of points, its longest prefix (which may be empty or all of S) that fits into a disk of radius r centered at a point of ℓ. In practice, $\texttt{prefix}_\ell(S, r)$ returns the length of the prefix, so 0 if it is empty.

2.3 Data structures in the plane

We now describe the data structures we employ to implement the above operations. Consider a polygonal curve P of length $n > 1$. For clarity of presentation we assume that n is a power of two; the general case requires minimal straightforward modifications. We denote by $P[i, j]$ the (contiguous) subcurve $(p_i, p_{i+1}, \ldots, p_j)$, for $1 \leq i \leq j \leq n$.

The hierarchical decomposition. The *hierarchical decomposition* of P is a binary tree, in which the root corresponds to $P = P[1, n]$, its left and right children to $P[1, n/2]$ and $P[n/2 + 1, n]$, respectively, and so forth. The leaves correspond to the single-vertex curves. Each node of the tree corresponds to a *canonical subcurve* of P (we will use the term "canonical set" when the ordering of the vertices in the subcurve is immaterial). A general subcurve $P[i, j]$ corresponds to the concatenation of $O(\log n)$ canonical subcurves, which can be identified from the value of the indices i and j in $O(\log n)$ time.

We often build data structures based on the hierarchical decomposition of a curve, where at each node we store an additional structure tailored to its corresponding canonical subcurve.

Farthest-point Voronoi diagram as a map. For a given set of points Q in the plane, the *farthest-point Voronoi diagram* $\text{FVD}(Q)$ is a planar map decomposing the plane into convex regions $R(p)$, one for each $p \in Q$, where $R(p)$ is the locus of all points in the plane for which p is the farthest point in Q. $\text{FVD}(Q)$ is a planar map with $O(|Q|)$ vertices, edges, and faces. Each face is a convex unbounded polygon. We preprocess the planar map for point location, so that, given a query point q, one can find the region $R(p)$ containing q, and thereby the farthest point of q in Q, in $O(\log |Q|)$ time.

Centroid decomposition. Consider a free tree T of degree at most three. A *centroid edge* is an edge of T whose removal splits T into two trees of almost equal size, with between $1/3$ and $2/3$ of the vertices of T. It is known to exist in any degree-at-most-three tree. The existence of a centroid edge naturally induces a *centroid decomposition* of T: the root is a centroid edge e. Each subtree left and right of the root is, recursively, a centroid decomposition of the two trees formed by the removal of e from T. Each internal node corresponds to an edge of T. Each leaf corresponds to either a single edge or to two edges of T.

Farthest-point Voronoi diagram as a tree. Given a set Q in the plane, the 1-skeleton of $\mathrm{FVD}(Q)$ (i.e., the collection of its vertices and edges, viewed as a graph with virtual vertices at the "ends" of infinite ray edges) is a tree, which, for Q in general position, has degree three. For simplicity of exposition, we will assume that the points of Q are in general position; refer to [5] for how to handle degeneracies in Q when constructing the centroid decomposition.

We now describe what is essentially the centroid-decomposition-based data structure from [5] for computing the smallest disk enclosing a given set Q of points, with the center on the query line ℓ, i.e., $\mathtt{disk}_\ell(Q)$. For a fixed set Q, the centroid decomposition of $\mathrm{FVD}(Q)$ requires linear space and $O(|Q|\log|Q|)$ time to build. In our data structure, for a path P, we build the centroid decomposition for all canonical subsets of P, which requires $O(n\log n)$ overall space and $O(n\log^2 n)$ time.

We store the centroid decomposition of $\mathrm{FVD}(Q)$ together with some additional geometric information, as follows: Each internal node x corresponds to an edge e_x of $\mathrm{FVD}(Q)$, which is a line segment[2] separating regions $R(v_x)$ and $R(w_x)$. Let $m(e_x)$ be the midpoint of e_x. There exist two infinite rays $\rho(v_x)$ and $\rho(w_x)$[5] emanating from $m(e_x)$, one fully contained in $R(v_x)$ and one in $R(w_x)$, respectively. Store these two rays together with the centroid edge e_x and points v_x and v_w at the current node x of the centroid decomposition tree. Notice that the polygonal line $\rho(v_x)\cup\rho(w_x)$ splits the plane into two regions: one contains the subtree corresponding to the left child of x and the other – the subtree of its right child.

The full structure T_{FVD}. Let $T_{\mathrm{FVD}} = T_{\mathrm{FVD}}(P)$ be the data structure constructed as follows: We start with the hierarchical decomposition of P. At each node of the decomposition, which corresponds to a canonical subcurve $P[i,j]$ of P, we store $\mathrm{FVD}(P[i,j])$ both in the planar map and centroid decomposition form.

2.4 Implementations

We now describe how to efficiently implement the primitives and analyze their running times.

2.4.1 Implementation of $\mathtt{disk}_\ell(Q)$

If Q is a canonical set of size n. The algorithm in [5] preprocesses Q in linear space and $O(n\log n)$ time to support such an operation in $O(\log n)$ time; see Section 2.3 for a description of the centroid decomposition data structure. Roughly speaking, the search proceeds by descending the centroid decomposition of $\mathrm{FVD}(Q)$, narrowing down the portion of the query line ℓ containing the disk center, at a constant cost per level. At a leaf of the decomposition, at most three farthest neighbors in Q remain, which allows to identify the center and radius directly. We use a more elaborate version below to solve the two-set version of the problem.

If Q is a union of two canonical sets. Let Q_1 and Q_2 be two canonical sets of total size at most n, and let ℓ be the query line. Put $Q := Q_1\cup Q_2$. We wish to compute $\mathtt{disk}_\ell(Q)$, the smallest disk with center on ℓ enclosing Q; let x^* denote its center, to be computed. We will need the following information about the sets Q_1 and Q_2 (already stored in T_{FVD}): $\mathrm{FVD}(Q_1)$ and $\mathrm{FVD}(Q_2)$ viewed both as planar maps, preprocessed for point location, and as planar trees, stored as centroid decompositions, see Section 2.3.

[2] Unbounded edges have to be handled slightly differently; we omit the details.

For clarity of explanation, identify ℓ with the x-axis and let x denote a generic point of ℓ. The center of the desired disk is the point $x = x^*$ minimizing $\text{radius}_\ell(Q, x) = \max(\text{radius}_\ell(Q_1, x), \text{radius}_\ell(Q_2, x))$. Each of the three functions are convex on ℓ, with a unique minimum, by Fact 1.

Note that the following *sidedness test* can be performed in $O(\log n)$ time: Given a point $x \in \ell$, we can determine whether x^* lies before, after, or at x along ℓ. Indeed, after querying with x in $\text{FVD}(Q_1)$ and $\text{FVD}(Q_2)$, we can determine the distance from x to the farthest points of Q (possibly more than one) and therefore the direction in which this distance decreases along ℓ – that's the direction towards x^*; if no such direction exists, then $x = x^*$. The bottleneck of this operation is the lookup of x in the planar map $\text{FVD}(Q_1)$ and $\text{FVD}(Q_2)$, at a cost of $O(\log n)$.

Finally, we describe the computation of $\texttt{disk}_\ell(Q)$. Throughout the search we maintain the current segment $s \subset \ell$ that is guaranteed to contain x^*. We initially set $s \coloneqq \ell$.

We now descend the centroid decompositions of $\text{FVD}(Q_1)$ and $\text{FVD}(Q_2)$. Without loss of generality, suppose we are currently descending the decomposition of $\text{FVD}(Q_1)$, with the current node corresponding to edge e and its two rays ρ_1, ρ_2 splitting the plane into wedges W_1 and W_2; s intersects $\rho_1 \cup \rho_2$ at most twice. At each of the intersection points we perform our sidedness test and eliminate part of s as the possible position of x^*. The result is an, in general, smaller updated segment s, contained fully in W_1 or in W_2 and containing x^*. We descend the decomposition of $\text{FVD}(Q_1)$ to the corresponding child of e.

Before terminating the current step, we check if the updated s crosses e. If it does, we split s at the intersection point, check what side contains x^* by another sidedness test, and shrink s further. Notice that the resulting segment s is guaranteed not to meet any of the edges of the discarded child of e in the centroid decomposition, nor e itself. Thus we inductively maintain the invariant that the current segment s can only intersect the edges of $\text{FVD}(Q_i)$ corresponding to the current subtree in its centroid decomposition.

We continue in this manner, until we reach the bottom of both hierarchies. At a single-edge leaf (a two-edge leaf is handled similarly) of a centroid decomposition of, say, $\text{FVD}(Q_1)$, one edge remains, and we apply the same trick to shrink the segment s to only one side of the edge, belonging to, say the Voronoi region of $q_1 \in Q_1$. Similarly, we narrow things down to one site $q_2 \in Q_2$. By construction, now $x^* \in s$, and s intersects no edges of $\text{FVD}(Q_1)$ nor of $\text{FVD}(Q_2)$. In particular, for all points on s, q_1 is the furthest point of Q_1 and q_2 – of Q_2. We check one of three possibilities: (a) q_1 could be the only point determining $\texttt{disk}_\ell(Q)$: we project q_1 orthogonally to ℓ and check if its projection q_1' lies in s. If so, check that Q is contained in the disk of radius $d(q_1, q_1')$ centered at q_1' (i.e., that $d(q_2, q_1') \leq d(q_1, q_1')$) – then this is $\texttt{disk}_\ell(Q)$. (b) We check if q_2 is the only point defining $\texttt{disk}_\ell(Q)$ similarly. Finally, (c) we construct the intersection of the bisector $b(q_1, q_2)$ with s (or, equivalently, with ℓ). This point q is the center of $\texttt{disk}_\ell(Q)$ and its radius is $d(q, q_1) = d(q, q_2)$. It is easily checked that the processing at the bottom of the two hierarchies takes $O(1)$ work once the identity of q_1 and q_2 is known.

Since the descent of the two logarithmic-height hierarchies takes $O(\log n)$ time for one step (the bottleneck in the sidedness test is consulting a point with known position in one of the FVDs for its furthest point in the other set: in each sidedness test we already know where we are in one diagram, but not in the other), we reach the bottom and the solution in $O(\log^2 n)$ time. We conclude that

▶ **Lemma 2.** *The operation* $\texttt{disk}_\ell(Q_1 \cup Q_2)$*, where* Q_1, Q_2 *are two canonical sets of total size* n*, can be performed in* $O(\log^2 n)$ *time.*

If Q is a union of $m \geq 2$ canonical sets.

▶ **Lemma 3.** *Let $Q_1, \ldots, Q_m$ be $m \geq 2$ canonical subsets of P of total size at most n, let their union be Q, and let ℓ be a line. We can compute $\texttt{disk}_\ell(Q)$ in $O(m^2 \log^2 n)$ time.*

Proof. For each pair (i, j), with $1 \leq i < j \leq m$, we compute $\texttt{disk}_\ell(Q_i \cup Q_j)$ by an application of the two-set procedure above and return the largest disk. The runtime bound is immediate, so we focus on correctness.

Since $\texttt{disk}_\ell(Q)$ is determined by at most two points of Q (refer to [16, Observation 1]), it follows that the desired disk is indeed $\texttt{disk}_\ell(Q_k \cup Q_l)$, if one of the defining points comes from Q_k and the other from Q_l with $l \neq k$; if the two points come from the same set Q_k, then for every $l \neq k$, $\texttt{disk}_\ell(Q_k \cup Q_l) = \texttt{disk}_\ell(Q)$. If $m = 2$, we are done. Otherwise, notice that only the largest of the disks $\texttt{disk}_\ell(Q_i \cup Q_j)$, $i < j$, can be $\texttt{disk}_\ell(Q)$, as any smaller disk would not enclose $Q_k \cup Q_l$, by definition of $\texttt{disk}_\ell(Q_k \cup Q_l)$ and Fact 1. ◀

▶ Remark 4. Lemma 3 is interesting in its own right, since it allows us to solve problems such as the following. Let S be a set of n points in the plane. Construct a data structure of near-linear size to support, in $O(\text{polylog}\, n)$ time, queries of the form: Given an axis-aligned rectangle R and a line ℓ, return the smallest enclosing circle of $S \cap R$ with center on ℓ.

2.4.2 Implementation of $\texttt{feasible}_\ell(Q, r)$

If Q is a single canonical set. Let $s \coloneqq \texttt{feasible}_\ell(Q, r)$ be the desired locus of candidate centers $x \in \ell$ such that $\text{radius}_\ell(Q, x) \leq r$; it is a contiguous interval on ℓ, if non-empty, since $\text{radius}_\ell(Q, x)$ is convex; see Fact 1. This interval is empty if $r < \text{radius}_\ell(Q)$, consists of a single point if $r = \text{radius}_\ell(Q)$, and is a positive-length interval with two endpoints if $r > \text{radius}_\ell(Q)$. Since $\text{radius}_\ell(Q, x)$ is convex and monotone along each of the two half-lines into which l is split by $\text{center}_\ell(Q)$, we can binary search along each half-line to determine the endpoints of s. Using the same data structure as in [5] allows us to conduct the binary search in $O(\log n)$ time. We omit the easy details.

If Q is the union of m canonical sets $Q_1, \ldots, Q_m$. Compute $s_i \coloneqq \texttt{feasible}_\ell(Q_i, r)$, for $i = 1, \ldots, m$, and return $\bigcap s_i$. Running time is dominated by the cost of computing the m feasible intervals, so it is $O(m \log n)$.

If Q is the union of a dynamic collection of canonical sets. The goal is to maintain the feasibility interval subject to addition and removal of canonical sets. For each set, store its feasible interval and, using two heaps H_L and H_R, maintain the rightmost left endpoint and leftmost right endpoint of the intervals. (i) On insertion of a new set Q', compute $[a_i, b_i] \coloneqq \texttt{feasible}_\ell(Q', r)$, add a_i to H_L, and b_i to H_R. (ii) On deletion of Q', remove the endpoints of the stored interval $\texttt{feasible}_\ell(Q', r)$ from H_L and H_R. (iii) The current feasible interval is always $[\max(H_L), \min(H_R)]$, unless $\max(H_L) > \min(H_R)$, in which case it is empty.

If we use a standard binary heap to implement H_L and H_R, insertion costs $O(\log n + \log m) = O(\log n)$, deletion $O(\log m)$, and current feasible region is available in time $O(1)$, where m is the current number of sets in Q and n is the total number of points involved.

2.4.3 Implementation of $\texttt{fits?}_\ell(Q, r)$

This test is logically equivalent to $\texttt{feasible}_\ell(Q, r) \neq \emptyset$ or to $\text{radius}_\ell(Q) \leq r$. The former is more efficient unless Q is a single canonical set (in which case the latter is more straightforward, but equally efficient asymptotically).

2.4.4 Implementation of $\texttt{prefix}_\ell(S, r)$

This operation can be implemented as a binary search along S, using $\texttt{fits?}(S', r)$ as a blackbox for candidate prefixes S' as the decision procedure, at the cost of a multiplicative logarithmic factor in running time. However, sometimes we can do better, as detailed below.

If S is a canonical subcurve. At any moment during a binary search in the hierarchical decomposition of S (which itself is a canonical subcurve of P), there are $O(\log n)$ canonical sets $S_1, \ldots, S_i$ comprising the current prefix being tested. As binary search progresses down the hierarchy, either a new canonical set S_{i+1} is added to the current prefix (if the prefix is too short), or S_i is replaced by a new, smaller S_i' (if the prefix is too long). We use the dynamic structure for feasible interval maintenance from Section 2.4.2 to keep track of this information and check if the current feasible interval is empty. Since the total number of insertions into and deletions from our collection of canonical sets is $O(\log n)$, the overall cost is $O(\log^2 n)$.

If S is a subcurve of P. Using the hierarchical decomposition of P, we express S as a concatenation $S_1 \cdot \ldots \cdot S_m$ of $m = O(\log n)$ canonical subcurves $S_1, \ldots, S_m$. In $O(m \log n)$ time, we compute the feasible intervals $s_1, ..., s_m \subset \ell$ for all subcurves S_i. Then in time $O(m)$, we compute the largest index j such that $S_1 \cdot \ldots \cdot S_j$ fits into a disk of radius r, but $S_1 \cdot \ldots \cdot S_{j+1}$ does not, by computing the largest $j \leq m$ such that $\bigcap_{1 \leq i \leq j} s_i \neq \emptyset$ (if $j = m$, $\texttt{prefix}_\ell(S, r)$ returns all of S). We then binary search within S_{j+1} to identify the longest prefix of S_{j+1} that can be concatenated to $S_i \cdot \ldots \cdot S_j$ and still fit into a disk of radius r in $O(\log^2 n)$ time, using a slight modification of the above single-canonical-subcurve algorithm. The overall time complexity is $O(m \log n + m + \log^2 n) = O(\log^2 n)$.

3 Discrete Fréchet distance simplification on a line

In this section we present an algorithm for answering queries, which refer to the already preprocessed input curve $P = (p_1, \ldots, p_n)$, of the following type: Given a line ℓ and an integer $k \geq 1$ find an at-most-k-vertex curve Q on ℓ minimizing $d_{\text{dF}}(P, Q)$; that is, find a curve $Q = (q_1, \ldots, q_{k'})$, where $k' \leq k$ and $q_1, \ldots, q_{k'} \in \ell$, that realizes the expression

$$\min_{\substack{k' \in [1,k] \\ q_1, \ldots, q_{k'} \in \ell}} \{d_{\text{dF}}(P, (q_1, \ldots, q_{k'}))\}.$$

We assume that $k < n$, since otherwise the curve $Q = (\overline{p}_1, \ldots, \overline{p}_n)$ clearly achieves the minimum, where $\overline{p}_i$ is the orthogonal projection of p_i on ℓ.

Recall that we write $P[i, j]$, for $1 \leq i \leq j \leq n$, to denote the (contiguous) subcurve $(p_i, \ldots, p_j)$ of P. For a point q in the plane, the distance from q to the vertex of $P[i, j]$ farthest from (nearest to) it, is denoted by $d_{\max}(P[i, j], q)$ ($d_{\min}(P[i, j], q)$). Notice that

$$\min_{q \in \ell} \{d_{\text{dF}}(P[i, j], (q))\} = \min_{q \in \ell} \{d_{\max}(P[i, j], q)\} = \text{radius}_\ell(P[i, j]),$$

where $\text{radius}_\ell(P[i, j])$ is the radius of $\texttt{disk}_\ell(P[i, j])$ (see Section 2.2).

Let $Q = (q_1, \ldots, q_{k'})$ be the desired curve and let W be a walk of P and Q of cost $d_{\mathrm{dF}}(P, Q)$. We may assume that W does not match two consecutive points in Q to the same point in P. That is, W does not contain two consecutive pairs of the form $(p_i, q_j), (p_i, q_{j+1})$, since, if it does, we may delete the pair (p_i, q_{j+1}) from W (and the point q_{j+1} from Q, if it was only matched to p_i) to obtain a legal walk W' (and a curve Q') of the same cost.

This assumption implies that, in order to compute $d_{\mathrm{dF}}(P, Q)$, we need to find an optimal partition of P into k' subcurves. That is,

$$d_{\mathrm{dF}}(P, (q_1, \ldots, q_{k'})) = \min_{1 \le i_1 < i_2 < \cdots < i_{k'-1} < n} \max \left\{ \begin{array}{c} d_{\max}(P[1, i_1], q_1) \\ \vdots \\ d_{\max}(P[i_{k'-2} + 1, i_{k'-1}], q_{k'-1}) \\ d_{\max}(P[i_{k'-1} + 1, n], q_{k'}) \end{array} \right\}.$$

We conclude that, given ℓ and k, our task is to find $k' \le k$ and a sequence of indices $1 \le i_1 < i_2 < \cdots < i_{k'-1} < n$ that minimize the expression

$$\max\{\mathrm{radius}_\ell(P[1, i_1]), \ldots, \mathrm{radius}_\ell(P[i_{k'-2} + 1, i_{k'-1}]), \mathrm{radius}_\ell(P[i_{k'-1} + 1, n])\}.$$

The desired sequence is then $Q = (\mathrm{center}_\ell(P[1, i_1]), \ldots, \mathrm{center}_\ell(P[i_{k'-1} + 1, n]))$, where $\mathrm{center}_\ell(P[i, j])$ is the center of $\mathtt{disk}_\ell(P[i, j])$.

3.1 The Algorithm

Given the supporting data structures and set of primitive operations, the query algorithm is pretty simple. We first present an algorithm for the decision problem, i.e., given ℓ, k, and r, determine if there exists an at-most-k-vertex curve Q on ℓ such that $d_{\mathrm{dF}}(P, Q) \le r$.

The decision algorithm. The function DECISION in Algorithm 1 uses a simple greedy recursive approach for solving the decision problem.

Algorithm 1 The decision algorithm: Given a distance $r > 0$, determine if there exists a sequence of at most k points on line ℓ at discrete Fréchet distance at most r from $P[i, n]$.

```
function DECISION(i, ℓ, k, r)
    if i = n + 1 then
        return TRUE
    if k = 0 then
        return FALSE
    l ← prefix_ℓ(P[i, n], r)
    if l = 0 then                                   ▷ p_i is too far from ℓ
        return FALSE
    return DECISION(i + l, ℓ, k − 1, r)
```

▶ **Lemma 5.** *Let ℓ be a line in the plane, $k \ge 1$ an integer, and $r > 0$. The call* DECISION$(1, \ell, k, r)$ *returns* TRUE *if and only if there exist $k' \le k$ points on ℓ, $Q = (q_1, \ldots, q_{k'})$, such that $d_{\mathrm{dF}}(P, Q) \le r$. The running time is k' times the cost of a call to $\mathtt{prefix}_\ell$ with a subcurve of P, i.e., $O(k \log^2 n)$.*

Proof. The time bound is obvious. Moreover, it is clear that, if the call DECISION$(1, \ell, k, r)$ returns TRUE, then there exist $k' \le k$ such points – namely, the centers of the disks corresponding to the computed prefixes. We now prove the other direction by induction on k.

If there exists a point q_1 on ℓ such that $d_{\mathrm{dF}}(P,(q_1)) \le r$, then the call $\texttt{prefix}_\ell(P[1,n],r)$ at the top level of the recursion will return n and the algorithm will return TRUE just at the beginning of the next level (since $i = n+1$).

Assume now that for any preprocessed curve P', if there exist fewer than k points on ℓ, $Q' = (q'_1,\ldots,q'_{k-1})$, such that $d_{\mathrm{dF}}(P',Q') \le r$, then the call DECISION$(1,\ell,k-1,r)$ (applied to P') returns TRUE. Moreover, assume there exist k points on ℓ, $Q = (q_1,\ldots,q_k)$, such that $d_{\mathrm{dF}}(P,Q) \le r$. We need to show that the call DECISION$(1,\ell,k,r)$ returns TRUE. Indeed, consider a walk W of P and Q of cost $d_{\mathrm{dF}}(P,Q)$, and let l' be the largest index such that W matches q_1 to $p_{l'}$. By definition, the index l returned by the call $\texttt{prefix}_\ell(P[1,n],r)$ at the top level of the recursion is at least as large as l'. Now, since the cost of W is $d_{\mathrm{dF}}(P,Q)$ and W matches q_1 to the prefix $P[1,l']$, we know that $d_{\mathrm{dF}}(P[l'+1,n],(q_2,\ldots,q_k)) \le r$ and therefore there exists a sequence Q' of at most $k-1$ points on ℓ, which is $(q_2,\ldots,q_k)$ or a suffix of it, such that $d_{\mathrm{dF}}(P[l+1,n],Q') \le r$. Therefore, by the induction hypothesis, the call DECISION$(l+i,\ell,k-1,r)$ at the top level of the recursion (where $i=1$) returns TRUE, and thus DECISION$(1,\ell,k,r)$ returns TRUE. ◀

▶ Note. A minor modification of the DECISION function, with the same preprocessing of P, can solve the problem of finding, given line ℓ and distance r, the shortest sequence of points Q on ℓ lying within discrete Fréchet distance r of P. (No such set exists if r is smaller than the maximum distance from a point of P to ℓ; this can be efficiently checked with no asymptotic slowdown.) The query can be answered in time $O(k\log^2 n)$, where n is the length of P and k is the length of the shortest such curve Q. Performing the same query on a subcurve $P[i,j]$ of P can be done at the same asymptotic cost.

The optimization algorithm. We now present the query algorithm. That is, given ℓ and k, find an at-most-k-vertex curve Q on ℓ minimizing $d_{\mathrm{dF}}(P,Q)$. The function OPTIMIZATION in Algorithm 2 actually returns the distance $d_{\mathrm{dF}}(P,Q)$, but it can be easily modified (within the same bounds) to return Q as well.

Algorithm 2 The optimization algorithm: Find the smallest distance r, for which there exists a sequence Q of at most k points on ℓ such that $d_{\mathrm{dF}}(P[i,n],Q) \le r$.

```
function OPTIMIZATION(i, ℓ, k)
    if k = 0 then return ∞
    Binary search for the smallest index j, i ≤ j ≤ n, such that
        DECISION(i, ℓ, k, radius_ℓ(P[i, j])) returns TRUE
    d_1 ← radius_ℓ(P[i, j])
    if j > i then
        d_2 ← OPTIMIZATION(j, ℓ, k − 1)
    else
        d_2 ← ∞
    return min{d_1, d_2}
```

▶ **Lemma 6.** *Let ℓ be a line in the plane, and let $k \ge 1$. The call OPTIMIZATION(i,ℓ,k) returns the smallest r, for which there exists a sequence Q of at most k points on ℓ such that $d_{\mathrm{dF}}(P,Q) \le r$. The running time is $O(k^2\log^3 n + k\log^4 n)$.*

Proof. If $k=1$, then the algorithm returns $\min_{q_1\in\ell} d_{\mathrm{dF}}(P[i,n],(q_1)) = \mathrm{radius}_\ell(P[i,n])$. Indeed, let j be the smallest index such that DECISION$(i,\ell,1,\mathrm{radius}_\ell(P[i,j]))$ returns TRUE, so the algorithm sets $d_1 \leftarrow \mathrm{radius}_\ell(P[i,j]) \le \mathrm{radius}_\ell(P[i,n])$. But since $k=1$, the latter inequality

must be an equality, so $d_1 = \text{radius}_\ell(P[i,n])$. As for d_2, it is set to ∞ either by the recursive call with $k = 0$ (if $j > i$) or by the **else** clause (if $j = i$). Hence, the algorithm returns $d_1 = \min_{q_1 \in \ell} d_{\text{dF}}(P[i,n],(q_1))$ as claimed.

Assume now that $k > 1$ and that the algorithm returns the correct value for $k-1$. Let

$$d \coloneqq \min_{\substack{k' \in [1,k] \\ q_1,\ldots,q_{k'} \in \ell}} d_{\text{dF}}(P[i,n],(q_1,\ldots,q_{k'})),$$

and let $Q = (q_1,\ldots,q_{k'})$ be a sequence of k' points on ℓ such that $d_{\text{dF}}(P[i,n],Q) = d$. Finally, let W be a minimum-cost walk of P and Q (i.e., a walk of cost d). Let $P[i,j]$ be the shortest prefix for which $\textsc{Decision}(i,\ell,k,\text{radius}_\ell(P[i,j]))$ returns TRUE. Then $d_1 = \text{radius}_\ell(P[i,j])$ and by definition $d_1 \geq d$. Notice that the inductive assumption implies that the value d_2 returned by the recursive call (if such a call is needed) is correct. We consider the cases $j = i$ and $j > i$ separately.

If $j = i$, $\text{radius}_\ell((p_i))$ is feasible. This radius is equal to the distance from p_i to ℓ. Notice that, for each point p of $P[i,n]$, the distance between p and ℓ is a lower bound on $d = \text{radius}_\ell(P[i,n])$, so we must have $d_1 = d$. Moreover, since $j \not> i$, the algorithm sets d_2 to ∞ and therefore returns the correct value.

Assume therefore that $j > i$ and let $d' \coloneqq \text{radius}_\ell(P[i,j-1])$. We know that the call $\textsc{Decision}(i,\ell,k,\text{radius}_\ell(P[i,j-1]))$ returned FALSE, so $d' < d \leq d_1$. We distinguish between two subcases.

- If $d_1 = d$, we claim that the value d_2 returned by the recursive call must be greater or equal than d_1, and therefore the algorithm returns the correct value, d_1. Indeed, if $d_2 < d_1$, then we found $k' \leq k$ points $q'_1,\ldots,q'_{k'}$, where $q'_1 = \text{center}_\ell(P[i,j-1])$ and $q'_2,\ldots,q'_{k'}$ are the centers induced by the recursive call, such that $d_{\text{dF}}(P[i,n],(q'_1,\ldots,q'_{k'})) \leq \max\{d',d_2\} < d_1$, contradicting the assumption that $d_1 = d = \min_{\substack{k' \in [1,k] \\ q_1,\ldots,q_{k'} \in \ell}} d_{\text{dF}}(P[i,n],(q_1,\ldots,q_{k'}))$.
- If $d_1 > d$, then let $P[i,j']$ be the prefix assigned to q_1 by the minimum-cost walk W. Notice that $j' \leq j-1$, since otherwise d would be greater or equal than d_1. So d (which as we recall is greater than d') is the smallest distance between $P[j'+1,n]$ and a curve on ℓ of length at most $k-1$, while d_2 is the smallest distance of either $P[j'+1,n]$ (if $j' = j-1$) or a proper suffix of $P[j'+1,n]$ (if $j' < j-1$) and such a curve on ℓ. Thus, $d \geq d_2$. But, since $\max\{d',d_2\} = d_2$ is feasible, we conclude that $d = d_2$ and the algorithm returns the correct value (since $d_2 = d < d_1$).

As for the running time, at each level of the recursion we issue $O(\log n)$ calls to the decision algorithm, where each of them is preceded by a call to $\texttt{disk}_\ell(P[i,j])$, with the appropriate index j, to obtain $\text{radius}_\ell(P[i,j])$. Since the running time of the decision algorithm is $O(k\log^2 n)$ and the running time of $\texttt{disk}_\ell(P[i,j])$ is $O(\log^4 n)$, and the number of levels is $O(k)$, we conclude that the total running time is $O(k^2\log^3 n + k\log^5 n)$.

This bound can be slightly improved by splitting the search into two stages: First find the canonical subcurve S_t that contains the point p_j that we are looking for, by adding subcurves one by one. Then, look for p_j within S_t by binary search using the tree of the canonical subcurve S_t. The search consists of $\log n$ steps, but in each step the set of canonical subcurves to which we need to apply $\texttt{disk}_\ell$ either grows by one, or we replace the last subcurve with a shorter one. Hence, $\texttt{disk}_\ell$ does not need to recompute the smallest disk for all $O(m^2)$ pairs of subcurves, but only for the $O(m)$ pairs involving the new curve. This effectively speeds up $\texttt{disk}_\ell$ by a logarithmic factor, resulting in total query time of $O(k^2\log^3 n + k\log^4 n)$. ◀

The following theorem summarizes our main result.

▶ **Theorem 7.** *Let P be a polygonal curve with n vertices in the plane. One can preprocess P in time $O(n \log^2 n)$ into a data structure of size $O(n \log n)$, so that given a line ℓ and a positive integer k, an at-most-k-vertex curve on ℓ minimizing $d_{\mathrm{dF}}(P, Q)$ (and the corresponding distance) can be found in $O(k^2 \log^3 n + k \log^4 n)$ time.*

4 Extension to geometric trees

A geometric tree is a tree whose vertices are points in the plane and whose edges are line segments connecting the corresponding points. In this section, we expand the data structure to support preprocessing a geometric tree T so that, given two vertices u, v of the tree, an arbitrary line ℓ, and an integer $k > 0$, one can find an at-most-k-vertex curve Q on ℓ that minimizes the Fréchet distance to the path Π_{uv} from u to v in T. We use ideas similar to those in [3], which in turn use heavy-path decomposition of T [17].

More specifically, the heavy-path decomposition of T is a collection $\{\pi_i\}$ of vertex-disjoint paths in T which collectively cover all the vertices of T. Any path Π_{uv} in T is a concatenation of $O(\log n)$ subpaths of (some of) the paths π_i and, this information can be extracted from the heavy-path decomposition data structure in time $O(\log^2 n)$ [17] (in fact, a faster implementation is possible, but would not help us, as it is not the bottleneck here). We then construct the data structure $T_{\mathrm{FVD}}(\pi_i)$, for each curve π_i, as above. Since the paths π_i are disjoint, the total size of the data structures involved is $O(n \log n)$, where n is the number of vertices in T.

Given a decomposition of Π_{uv} as above, we simply run Algorithm 2 (which, in turn, invokes Algorithm 1), on a concatenation of $l = O(\log n)$ subpaths $P_1, \ldots, P_l$ of preprocessed heavy paths. Decision algorithm requires an implementation of $\texttt{prefix}_\ell$ for such a concatenation. Since we can partition the subpaths further, each into $O(\log n)$ canonical subpaths, we are effectively running the $\texttt{prefix}_\ell$ algorithm for a subpath, as in Section 2.4.4, but with $m = l \cdot O(\log n) = O(\log^2 n)$ canonical subpaths, resulting in running time of $O(\log^3 n)$ for $\texttt{prefix}_\ell$ and $O(k \log^3 n)$ for Decision.

As for Optimization, conceptually we run Algorithm 2 with minor modifications to speed it up. There are at most k levels of recursion. We will describe and analyze one level and multiply the runtime by k.

At a fixed level of recursion, we need to find the shortest prefix of the concatenation of $O(\log n)$ subpaths $P_1, P_2, \ldots, P_l$ that satisfies the Decision condition. We first find j so that $P_1 \cdot \ldots \cdot P_j$ satisfies it, but $P_1 \cdot \ldots \cdot P_{j-1}$ does not (or $j = 1$, in which case we are done), adding one subpath at a time, sequentially. This takes at most l invocations of Decision, for a total of $O(k \log^4 n)$ time. Now we subdivide P_j into a logarithmic number of canonical subcurves and apply the same logic to find the canonical subcurve P' that contains the (still unknown) point p_j we want. This involves $O(\log n)$ more invocations of Decision, for a total of $O(k \log^4 n)$ time.

However, we did not account for the work of the radius_ℓ algorithm. It is easily checked that during these first two steps, over all the calls to radius_ℓ, it is enough to compute the minimum enclosing radius with center on ℓ for at most $\binom{m}{2} = O(\log^4 n)$ pairs of canonical subpaths constituting the subpaths $P_1, \ldots, P_l$, at a cost of $O(\log^2 n)$ per pair; so the total cost of all the radius_ℓ runs at the current level of recursion up to now is at most $O(\log^6 n)$.

Finally, we use the hierarchical decomposition of P' to find desired point p_j, essentially as in the proof of Theorem 7. We observe that descending the hierarchy in binary search, we always either add the last canonical subpath or replace the last canonical subpath, so the number of pairs of subpaths that need to be recomputed by the radius_ℓ algorithm on each descent step is only $O(m) = O(\log^2 n)$ (rather than $O(m^2)$) and therefore each radius_ℓ call takes $O(m \log^2 n) = O(\log^4 n)$ time, for a total of $O(\log^5 n)$ over the entire binary search.

To summarize, each level of recursion in this implementation of Optimization takes $O(k \log^4 n + \log^6 n)$ time. We conclude that that the full running time of Optimization is $O(k^2 \log^4 n + k \log^6 n)$, which finishes the proof of the following theorem:

▶ **Theorem 8.** *Let T be a geometric tree on n vertices. One can preprocess T in time $O(n \log^2 n)$ into a data structure of size $O(n \log n)$, so that given vertices u and v in T, a line ℓ, and a positive integer k, the at-most-k-vertex curve Q on ℓ minimizing the discrete Fréchet distance between Q and the path Π_{uv} between u and v in T, can be found in $O(k^2 \log^4 n + k \log^6 n)$ time.*

5 Extension to polygonal regions in the plane

Let R be a (topologically closed) polygonal region in the plane bounded by m edges and P be an n-vertex curve. Given an integer $k > 0$, we show how to find an at-most-k-vertex curve $Q \subseteq R$ that minimizes the discrete Fréchet distance $d_{dF}(P, Q)$. That is, we find a curve $Q = (q_1, \ldots, q_{k'})$, with $k' \leq k$ and $q_1, \ldots, q_{k'} \in R$, that realizes the expression

$$\min_{\substack{k' \in [1,k] \\ q_1,\ldots,q_{k'} \in R}} \{d_{\mathrm{dF}}(P, (q_1, \ldots, q_{k'}))\}.$$

Note that Q need not be unique.

Observe that we only need to implement one new primitive, which we refer to as $\texttt{disk}_R(S)$, that computes the smallest-radius disk centered at a point of R for a subcurve S of P (and its corresponding radius and center). Using $\texttt{disk}_R(S)$ one can perform binary search over P to implement $\texttt{prefix}_R(S, r)$ at the cost of a logarithmic factor in the running time. The Decision and Optimization functions (Algorithms 1 and 2) will use the new primitive and otherwise remain the same.

We now sketch how to implement $\texttt{disk}_R(S)$ for a subcurve $S = P[i, j]$. Let c be the center of the mec containing the subcurve $P[i, j]$ and centered at a point of R.

First we find the center of the unconstrained mec of $P[i, j]$, using the data structure in [9] for the mec of a planar point set. We partition $P[i, j]$ into $t = O(\log n)$ canonical subcurves (as in Section 2.3) and preprocess each subcurve as in [9]. Using the algorithm from Section 3 of [9], we can compute the mec of the union of the subcurves, that is, of $P[i, j]$ in deterministic time $O(t^3 \log^3 n) = O(\log^6 n)$ (refer to the discussion after Lemma 2 in that paper). Alternatively, Theorem 2 in [9] provides an algorithm for finding the mec in expected time $O(t \log t \log n + \sqrt{t} \log t \log^3 n) = O(\log^{3.5} n \log \log n)$.[3] If the center of the resulting mec lies in R, we are done: we found $\texttt{disk}_R(S)$.

Otherwise, since $d_{\max}(P[i, j], x)$ is a convex function of x, the center c lies on the boundary of R. For each bounding segment s of R, we compute $\texttt{disk}_\ell(P[i, j])$ for the supporting line ℓ of s in time $O(t^2 \log^2 n) = O(\log^4 n)$; refer to Lemma 3. Once again, if the center $c(s) := \mathrm{center}_\ell(P[i, j])$ lies in s, this gives a candidate mec, otherwise the endpoint of s closest to $c(s)$ provides such a candidate. After repeating the process for each boundary edge of R, we return (the center of) the smallest disk found. The total cost is thus $O(m \log^4 n)$.

To summarize, $\texttt{disk}_R(P[i, j])$ can be computed in deterministic time $O(\log^6 n + m \log^4 n)$ or in expected time $O(\log^{3.5} n \log \log n + m \log^4 n) = O(m \log^4 n)$, assuming $m \neq 0$.

[3] The statements are phrased in terms of solving an LP over an intersection of polyhedra in $\mathbb{R}^3$, but a later discussion points out that the same logic, with minor modifications, applies to finding the mec of a discrete point set in the plane as well [9].

We now implement $\texttt{prefix}_R$ by binary search over the prefix length using $\texttt{disk}_R(S)$ in expected time $O(m \log^5 n)$. Expected running time of DECISION will be $O(km \log^5 n)$ and that of OPTIMIZATION will be $O(k^2 m \log^6 n)$ (see the time analysis of Lemma 6 for details).

We did not optimize the logarithmic factors.

▶ **Theorem 9.** *Let P be a polygonal curve with n vertices in the plane. One can preprocess P in time $O(n \log^2 n)$ into a data structure of size $O(n \log n)$, so that given a polygonal domain R bounded by m segments, an at-most-k-vertex curve $Q \subseteq R$ that minimizes the discrete Fréchet distance $d_{dF}(P, Q)$ and the corresponding distance can be found in expected time $O(k^2 m \log^6 n)$. Alternatively, it can be found in deterministic time $O(k^2 m \log^6 n + k^2 \log^8 n)$.*

6 Discrete Fréchet distance simplification on a g-flat

In this section we address the general version of the dimension-reduction problem mentioned in the introduction. That is, P is a polygonal curve in $\mathbb{R}^d$, for some constant $d > 2$, and we need to preprocess P into a compact data structure to efficiently support queries that specify a g-flat h ($1 \leq g \leq d-1$) and an integer $k \geq 1$, and request a curve Q on h of length at most k that minimizes the discrete Fréchet distance to P.

It is unlikely that there exists a solution to this problem with bounds similar to those that we obtained in Section 3, since it seems that, on the one hand, a primitive analogous to $\texttt{disk}_\ell(Q)$, where "disk" is replaced by "ball" and ℓ is replaced by h, is essential, but on the other hand, it is probably impossible to implement such a primitive efficiently in higher dimensions, i.e., in polylogarithmic time after near-linear time preprocessing of Q.

Therefore, since we are interested in a solution to the general dimension-reduction problem with near-linear time preprocessing and polylogarithmic time query, we resort to approximate queries. Specifically, we present such a solution (for a prespecified parameter $0 < \varepsilon_0 < 1$) that given h returns a curve Q on h of length at most k that minimizes the discrete Fréchet distance to P up to a factor of $1 + \varepsilon_0$ (i.e., $d_{\mathrm{dF}}(P, Q) \leq (1 + \varepsilon_0) d_{\mathrm{dF}}(P, Q^*)$, where Q^* is such a curve that minimizes the discrete Fréchet distance to P).

Set $\varepsilon := \frac{\varepsilon_0}{1+\varepsilon_0}$; then, $0 \leq \varepsilon \leq 1/2$ and $\frac{1}{1-\varepsilon} = 1 + \varepsilon_0$. We first define the primitive $\texttt{ball}_h^\varepsilon(Q)$, which returns a $(1-\varepsilon)$-approximation of the smallest ball centered at a point of h and containing a set Q of points. That is, let $\mathrm{radius}_h^\varepsilon(Q)$ denote the radius of the returned ball, then $(1-\varepsilon)\,\mathrm{radius}_h(Q) \leq \mathrm{radius}_h^\varepsilon(Q) \leq \mathrm{radius}_h(Q)$, where $\mathrm{radius}_h(Q)$ is the radius of the smallest ball with center on h.

We use *coresets* to implement $\texttt{ball}_h^\varepsilon(Q)$, assuming Q is the set of points corresponding to a subsequence of P. More precisely, we preprocess P so that given a subsequence $P[i, j]$, for $1 \leq i \leq j \leq n$, one can compute $\texttt{ball}_h^\varepsilon(P[i, j])$ in $O(c_\varepsilon \log n)$ time, where $c_\varepsilon = c_\varepsilon(\varepsilon)$ is a constant, see below. Before presenting the details, we provide a brief overview of the coreset-related definitions and results that we use.

Coresets. Let S be a compact set in $\mathbb{R}^d$. Let $\theta \in \mathbb{S}^{d-1}$ be a direction. The *directional width* $\mathrm{width}(S, \theta)$ of S in direction θ is the minimum distance between a pair of hyperplanes with normal θ containing S between them. Fix a number ε, $0 < \varepsilon < 1$. A subset C of S is an *ε-coreset for S with respect to directional width*, if it has the property that $\mathrm{width}(C, \theta) \geq (1-\varepsilon)\,\mathrm{width}(S, \theta)$, for all directions θ; since $C \subset S$, $\mathrm{width}(C, \theta) \leq \mathrm{width}(S, \theta)$. We say that a subset C of S is an *ε-coreset* for S for *farthest neighbors* if, for any point q, $d_{\max}(C, q) \geq (1-\varepsilon) d_{\max}(S, q)$. It follows that if B is any ball containing C, then the ball $\frac{B}{1-\varepsilon}$, obtained by scaling B around its center by a factor of $1/(1-\varepsilon)$, contains S.

The following facts are known (refer for example to [14, Section 23.1]):

- A compact set in $\mathbb{R}^d$ has an ε-coreset for directional width of size $c_\varepsilon := O(1/\varepsilon^{(d-1)/2})$.
- Such an ε-coreset for a set of n points can be constructed in time $O(n + c_\varepsilon^3)$.
- An $(\varepsilon/4)$-coreset for directional width is an ε-coreset for farthest neighbors.
- If $C_1, C_2, \ldots, C_t$ are ε-coresets for $S_1, S_2, \ldots, S_t$ for width or farthest neighbors, then $\bigcup C_i$ is an ε-coreset for $\bigcup S_i$ for width or farthest neighbors.

As we are interested in coresets for farthest neighbors, with a slight abuse of terminology, we will use unqualified "coreset" to denote them hereafter.

Smallest ball with a center on a flat. We recall the classical randomized LP-type algorithm of Welzl [20] for finding the smallest enclosing ball of an n-point set S in $\mathbb{R}^d$ in $O_d(n)$ expected time (with the implied constant depending on d). An inspection of the algorithm reveals that it can be used essentially unmodified to compute the smallest enclosing ball of S whose center is constrained to lie in a given g-flat; such a ball will be defined by at most $g + 1$ points.

Implementation of $\texttt{ball}_h^\varepsilon(P[i,j])$. We now explain how to implement $\texttt{ball}_h^\varepsilon(P[i,j])$, which is the basic building block of the approximation algorithm described above. As in Section 2.3 we once again construct a hierarchical decomposition of P, storing P of length n, then $P[1, n/2]$ and $P[n/2 + 1, n]$ as its children and so forth, thereby creating a hierarchy of canonical subcurves. With each canonical subcurve $P[i, j]$ we associate its coreset C_{ij} for farthest neighbors, of size at most c_ε. An arbitrary subcurve P' of P is the union of $O(\log n)$ canonical subcurves. If we collect the points of the corresponding coresets, we obtain a coreset C' for P' of size $c_\varepsilon \log n$. Running the exact minimum-enclosing-ball with center on a g-flat algorithm on the set C', by the above discussion, gives an ε-approximation of corresponding ball for P'. Indeed, we obtain the smallest possible ball B with the center constrained to the flat and enclosing C'. By the coreset property $\frac{B}{1-\varepsilon}$ contains P'. There cannot be a substantially smaller ball with center on the flat and containing P', as such a ball would also contain C' and would contradict minimality of B.

The expected running time is dominated by that of the ball-finding routine, which is $O_d(c_\varepsilon \log n) = O_d(\log n/\varepsilon^{(d-1)/2})$, concluding our description of an implementation of $\texttt{ball}_h^\varepsilon$.

The decision algorithm. We now wish to devise a decision algorithm, similar to the one in Section 3. Given $r > 0$, our decision algorithm will return TRUE if there exists a sequence Q of at most k points on h such that $d_{\mathrm{dF}}(P, Q) \leq \frac{r}{1-\varepsilon}$. It will return FALSE otherwise, in which case we may only conclude that for any sequence Q of at most k points on h, $d_{\mathrm{dF}}(P, Q) > r$.

For this purpose, we define the primitive $\texttt{prefix}_h^\epsilon(S, r)$ for a sequence S of points and a radius r. It returns a prefix $S[1, k]$ of S, such that $\texttt{ball}_h^\varepsilon(S[1, k])$ returns a radius that is at most r where as $\texttt{ball}_h^\varepsilon(S[1, k + 1])$ returns a radius that is greater than r. In practice, $\texttt{prefix}_h^\varepsilon(S, r)$ returns the length of the prefix, so 0 if it is empty. Thus, if $\texttt{prefix}_h^\varepsilon(S, r) = k$, $0 < k < n$, then $\mathrm{radius}_h(S[1, k]) \leq \frac{r}{1-\varepsilon}$ and $\mathrm{radius}_h(S[1, k + 1]) > r$. (If $\texttt{prefix}_h^\varepsilon(S, r) = 0$, then $\mathrm{radius}_h(S[1, 1]) > r$, and if $\texttt{prefix}_h^\varepsilon(S, r) = n$, then $\mathrm{radius}_h(S[1, n]) \leq \frac{r}{1-\varepsilon}$.)

▶ **Observation 10.** *If* $\texttt{prefix}_h(S, r)$ *returns the largest* k' *for which* $\mathrm{radius}_h(S[1, k']) \leq r$, *then* $\texttt{prefix}_h(S, r) \leq \texttt{prefix}_h^\varepsilon(S, r) \leq \texttt{prefix}_h(S, \frac{r}{1-\varepsilon})$.

To implement $\texttt{prefix}_h^\epsilon(S, r)$ we perform a binary search. That is, we set $i = \lfloor \frac{|S|}{2} \rfloor$ and call $\texttt{ball}_h^\varepsilon(S[1, i])$. Now, if $\mathrm{radius}_h^\varepsilon(S[1, i]) \leq r$, then we increase i, and otherwise we decrease i, etc. Thus, the cost of a call to $\texttt{prefix}_h^\epsilon$ with a subcurve of P is $O(c_\varepsilon \log^2 n)$ time.

We now present the decision algorithm (see Algorithm 3).

Algorithm 3 The decision algorithm: Given a distance $r > 0$, determine if there exists a sequence of at most k points on h at discrete Fréchet distance at most $\frac{r}{1-\varepsilon}$ from $P[i, n]$.

```
function ApproxDecision(i, h, k, r)
    if i = n + 1 then
        return TRUE
    if k = 0 then
        return FALSE
    l ← prefix^ε_h(P[i, n], r)
    if l = 0 then                                   ▷ p_i is too far from h
        return FALSE
    return ApproxDecision(i + l, h, k − 1, r)
```

The following lemma follows from Observation 10.

▶ **Lemma 11.** *Let h be a g-dimensional flat in $\mathbb{R}^d$, $k \geq 1$ an integer, and $r > 0$. If the call* ApproxDecision$(1, h, k, r)$ *returns* TRUE*, then there exists a sequence Q of $k' \leq k$ points on h, such that $d_{\mathrm{dF}}(P, Q) \leq \frac{r}{1-\varepsilon}$. If it returns* FALSE*, then for any sequence Q of $k' \leq k$ points on h, $d_{\mathrm{dF}}(P, Q) > r$. The running time is k' times the cost of a call to* $\texttt{prefix}^{\varepsilon}_h$ *with a subcurve of P.*

Finally, we run the optimization algorithm in Section 3, using the decision algorithm above (and replacing radius_ℓ by $\mathrm{radius}^{\varepsilon}_h$). The following theorem summarizes the section's main result.

▶ **Theorem 12.** *Let P be a polygonal curve with n vertices in $\mathbb{R}^d$. Let $0 < \varepsilon_0 < 1$, and set $\varepsilon = \frac{\varepsilon_0}{1+\varepsilon_0}$. One can preprocess P in time $O(n \log n + \frac{n}{\varepsilon^{3(d-1)/2}})$ into a data structure of size $O(n \log n + \frac{n}{\varepsilon^{(d-1)/2}})$, so that given a g-dimensional flat h and a positive integer k, an at-most-k-vertex curve Q on h such that $d_{\mathrm{dF}}(P, Q) \leq \frac{d_{\mathrm{dF}}(P,Q^*)}{1-\varepsilon} = (1 + \varepsilon_0) d_{\mathrm{dF}}(P, Q^*)$ (and the corresponding distance) can be found in $O(\frac{k^2 \log^2 n}{\varepsilon^{(d-1)/2}})$ time, where Q^* is such a curve that minimizes the discrete Fréchet distance to P.*

7 Discussion and open problems

We mention several open problems:

- We did not strive to optimize the query time. How far can it be reduced, without substantially increasing the space and preprocessing time, in Theorems 7, 8, 9, and 12?
- Can the factor k^2 in the query time in these theorems be made linear in k? Or at least subquadratic?

References

1 Pankaj K Agarwal. Range searching. In *Handbook of Discrete and Computational Geometry*, pages 1057–1092. Chapman and Hall/CRC, 2017.

2 Pankaj K. Agarwal, Sariel Har-Peled, Nabil H. Mustafa, and Yusu Wang. Near-linear time approximation algorithms for curve simplification. *Algorithmica*, 42(3-4):203–219, 2005. doi:10.1007/S00453-005-1165-Y.

3 Boris Aronov, Tsuri Farhana, Matthew J. Katz, and Indu Ramesh. Discrete Fréchet distance oracles. In Wolfgang Mulzer and Jeff M. Phillips, editors, *40th International Symposium on Computational Geometry (SoCG 2024)*, pages 10:1–10:14, 2024. doi:10.4230/LIPIcs.SoCG.2024.10.

4 S. Bereg, M. Jiang, W. Wang, B. Yang, and B. Zhu. Simplifying 3D polygonal chains under the discrete Fréchet distance. In *LATIN 2008: Theoretical Informatics, 8th Latin American Symposium, Búzios, Brazil, April 7-11, 2008, Proceedings*, volume 4957 of *Lecture Notes in Computer Science*, pages 630–641. Springer, 2008. doi:10.1007/978-3-540-78773-0_54.

5 Prosenjit Bose, Stefan Langerman, and Sasanka Roy. Smallest enclosing circle centered on a query line segment. In *Proceedings of the 20th Annual Canadian Conference on Computational Geometry, Montréal, Canada, August 13-15, 2008*, 2008.

6 Peter Brass, Christian Knauer, Chan-Su Shin, Michiel H. M. Smid, and Ivo Vigan. Range-aggregate queries for geometric extent problems. In Anthony Wirth, editor, *Nineteenth Computing: The Australasian Theory Symposium, CATS 2013, Adelaide, Australia, February 2013*, volume 141 of *CRPIT*, pages 3–10. Australian Computer Society, 2013. URL: http://crpit.scem.westernsydney.edu.au/abstracts/CRPITV141Brass.html.

7 Karl Bringmann and Bhaskar Ray Chaudhury. Polyline simplification has cubic complexity. *J. Comput. Geom.*, 11(2):94–130, 2020. doi:10.20382/JOCG.V11I2A5.

8 A. Driemel and S. Har-Peled. Jaywalking your dog: Computing the Fréchet distance with shortcuts. *SIAM J. Comput.*, 42(5):1830–1866, 2013. doi:10.1137/120865112.

9 David Eppstein. Dynamic three-dimensional linear programming. *ORSA Journal on Computing*, 4(4):360–368, 1992. doi:10.1287/IJOC.4.4.360.

10 Chenglin Fan, Omrit Filtser, Matthew J. Katz, Tim Wylie, and Binhai Zhu. On the chain pair simplification problem. In Frank Dehne, Jörg-Rüdiger Sack, and Ulrike Stege, editors, *Algorithms and Data Structures - 14th International Symposium, WADS 2015, Victoria, BC, Canada, August 5-7, 2015. Proceedings*, volume 9214 of *Lecture Notes in Computer Science*, pages 351–362. Springer, 2015. doi:10.1007/978-3-319-21840-3_29.

11 O. Filtser. Universal approximate simplification under the discrete Fréchet distance. *Inf. Process. Lett.*, 132:22–27, 2018. doi:10.1016/j.ipl.2017.10.002.

12 Leonidas J. Guibas, John Hershberger, Joseph S. B. Mitchell, and Jack Snoeyink. Approximating polygons and subdivisions with minimum link paths. *Int. J. Comput. Geom. Appl.*, 3(4):383–415, 1993. doi:10.1142/S0218195993000257.

13 Prosenjit Gupta, Ravi Janardan, Yokesh Kumar, and Michiel H. M. Smid. Data structures for range-aggregate extent queries. *Comput. Geom.*, 47(2):329–347, 2014. doi:10.1016/J.COMGEO.2009.08.001.

14 Sariel Har-Peled. *Geometric approximation algorithms*. Number 173 in Mathematical Surveys and Monographs. American Mathematical Soc., 2011.

15 Arindam Karmakar, Sasanka Roy, and Sandip Das. Fast computation of smallest enclosing circle with center on a query line segment. *Inf. Process. Lett.*, 108(6):343–346, 2008. doi:10.1016/J.IPL.2008.07.002.

16 Sasanka Roy, Arindam Karmakar, Sandip Das, and Subhas C. Nandy. Constrained minimum enclosing circle with center on a query line segment. *Comput. Geom.*, 42(6-7):632–638, 2009. doi:10.1016/J.COMGEO.2009.01.002.

17 D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983. doi:10.1016/0022-0000(83)90006-5.

18 Mees van de Kerkhof, Irina Kostitsyna, Maarten Löffler, Majid Mirzanezhad, and Carola Wenk. Global curve simplification. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, volume 144 of *LIPIcs*, pages 67:1–67:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.ESA.2019.67.

19 Marc J. van Kreveld, Maarten Löffler, and Lionov Wiratma. On optimal polyline simplification using the Hausdorff and Fréchet distance. *J. Comput. Geom.*, 11(1):1–25, 2020. doi:10.20382/JOCG.V11I1A1.

20 Emo Welzl. Smallest enclosing disks (balls and ellipsoids). In Hermann A. Maurer, editor, *New Results and New Trends in Computer Science, Graz, Austria, June 20-21, 1991, Proceedings [on occasion of H. Maurer's 50th birthday]*, volume 555 of *Lecture Notes in Computer Science*, pages 359–370. Springer, 1991. doi:10.1007/BFB0038202.

Polynomial Equivalence of Extended Chemical Reaction Models

Divya Bajaj
University of Texas Rio Grande Valley, Edinburg, TX, USA

Jose-Luis Castellanos
University of Texas Rio Grande Valley, Edinburg, TX, USA

Ryan Knobel
University of Texas Rio Grande Valley, Edinburg, TX, USA

Austin Luchsinger
University of Texas Rio Grande Valley, Edinburg, TX, USA

Aiden Massie
University of Texas Rio Grande Valley, Edinburg, TX, USA

Adrian Salinas
University of Texas Rio Grande Valley, Edinburg, TX, USA

Pablo Santos
University of Texas Rio Grande Valley, Edinburg, TX, USA

Ramiro Santos
University of Texas Rio Grande Valley, Edinburg, TX, USA

Robert Schweller
University of Texas Rio Grande Valley, Edinburg, TX, USA

Tim Wylie
University of Texas Rio Grande Valley, Edinburg, TX, USA

Abstract

The ability to detect whether a species (or dimension) is zero in Chemical Reaction Networks (CRN), Vector Addition Systems, or Petri Nets is known to increase the power of these models – making them capable of universal computation. While this ability may appear in many forms, such as extending the models to allow transitions to be inhibited, prioritized, or synchronized, we present an extension that directly performs this zero checking. We introduce a new *void genesis* CRN variant with a simple design that merely increments the count of a specific species when any other species' count goes to zero. As with previous extensions, we show that the model is Turing Universal. We then analyze several other studied CRN variants and show that they are all equivalent through a *polynomial* simulation with the void genesis model, which does not merely follow from Turing-universality. Thus, inhibitor species, reactions that occur at different rates, being allowed to run reactions in parallel, or even being allowed to continually add more volume to the CRN, does not add additional *simulation power* beyond simply detecting if a species count becomes zero.

2012 ACM Subject Classification Theory of computation → Models of computation

Keywords and phrases Chemical Reaction Networks, Simulations, Petri-nets, Vector Addition Systems, Bi-simulation, Turing-universality, Inhibitors

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.7

Related Version *Full Version*: https://doi.org/10.48550/arXiv.2509.15584

Funding This research was supported in part by National Science Foundation Grant CCF-2329918.

1 Introduction

Background. Chemical Reaction Networks [5], Vector Addition Systems [18], and Petri nets [23] are three formalisms that arose in different disciplines to study distributed/concurrent systems. Despite their distinct origins, these models are mathematically equivalent in expressive power [8, 15]. While these models are capable of very complex behavior – e.g., deciding if one configuration is reachable from another via a sequence of transitions was recently shown to be Ackermann-complete [9, 19] – they fall short of Turing-universality.

© Divya Bajaj, Jose-Luis Castellanos, Ryan Knobel, Austin Luchsinger, Aiden Massie, Adrian Salinas, Pablo Santos, Ramiro Santos, Robert Schweller, and Tim Wylie;
licensed under Creative Commons License CC-BY 4.0

Even though these base models are not capable of universal computation, they are each right on the cusp of doing so. It is well known that extending the models in any way that allows "checking for zero" immediately results in Turing-universality [1, 16, 22]. This has been shown for model extensions allowing transition inhibition [1, 7, 11, 16], transition prioritization [16, 24, 25], synchronous (parallel) transitions [6, 28], or even continual volume increase [2, 3]. These results are typically established by showing how each extended model can simulate a register (counter) machine [22].

However, Turing equivalence alone is a blunt instrument. It tells us that models can emulate one another, but not how efficiently this can be done. We often care not just that two models are Turing-complete, but if they can be easily transformed into one another. For instance, Hack gave one such transformation between Inhibitory and Priority Petri nets [16], but discussed their equivalence in terms of languages. For this paper, we focus on the concept of simulation to draw comparisons between models.

In the literature, several notions of simulation have been proposed to capture structural or behavioral equivalence between systems. While concepts like strong/weak bisimulation [4, 10, 12, 17, 21] and pathway decomposition [26, 27] have been used to compare the expressiveness of different systems, there is typically a tradeoff between reasonably preserving dynamics and maintaining structural correspondence. Furthermore, efficiency of simulation is often either implicitly included or sometimes omitted altogether. Part of our work aims to introduce a more wieldy definition of simulation that explicitly accounts for efficiency. We give a more detailed discussion on simulation later in the paper.

Our Contributions. In this work, we make two main contributions:

1. We define a general-purpose notion of polynomial efficient simulation that is intended to capture a broader notion of simulation while still remaining reasonable. Our definition ensures that the simulation respects a polynomial correspondence in both time (dynamics) and space (structure). This allows us to formally compare the simulation efficiency of different CRN variants.
2. We introduce a new model, which we call Void Genesis (VG), that makes zero-testing explicit: it creates a designated species whenever a tracked species reaches zero. We use this model as a unifying model that captures the essence of zero-testing in a clean and modular way. We show that the Void Genesis model is polynomially equivalent to other known Turing-universal extensions of CRNs.

Since all of these extended models can be simulated by VG (and vice versa) with only polynomial overhead, our results establish a hub of polynomial simulation equivalence among them. This yields a stronger and more precise understanding of the relationships between these CRN variants and highlights VG simulation as a useful proof technique for adding models to this hub. Figure 1 summarizes the simulation relationships we establish here.

Organization. Given the number of models and results, we have arranged the paper in a way that systematically builds up understanding and techniques. Section 2 covers the formal definitions of the models and a simulation of one model with another. Section 3 provides minimum working examples in the models. The polynomial equivalence between models is proven through a series of simulations in Section 4. In Section 5, we discuss our results and some open problems.

2 Preliminaries

In Section 2.1, we define the extended chemical reaction network models considered in this paper with examples, and in Section 2.2 we define the concept of inter-model simulation.

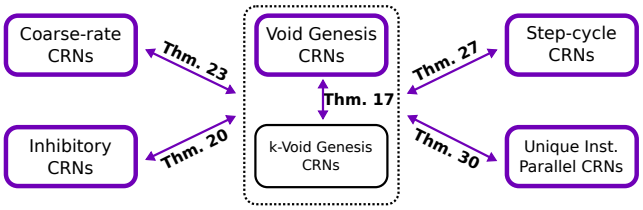


Figure 1 CRN Model variants and their connections to Void Genesis CRNs. Theorems 1-5 each show a polynomial equivalence between two CRN variants. With the Void Genesis model as a central hub, the implication of these results is a polynomial equivalence between all models.

2.1 Models

Here, we define the six chemical reaction networks considered in this paper: the basic chemical reaction network model (Section 2.1.1), the Void Genesis model (Section 2.1.2), the Inhibitory CRN model (Section 2.1.3), the Coarse-Rate CRN model (Section 2.1.4), the Step-Cycle CRN model (Section 2.1.5), and the Unique Instruction Parallel model (Section 2.1.6).

2.1.1 Chemical Reaction Networks

A *chemical reaction network* (CRN) $\mathcal{C} = (\Lambda, \Gamma)$ is defined by a finite set of species Λ, and a finite set of reactions Γ where each reaction is a pair $(\vec{R}, \vec{P}) \in \mathbb{N}^\Lambda \times \mathbb{N}^\Lambda$, sometimes written $\vec{R} \longrightarrow \vec{P}$, that denotes the *reactant* species consumed by the reaction and the *product* species generated by the reaction. For example, given $\Lambda = \{a, b, c\}$, the reaction $((2,0,0),(0,1,1))$ represents $2a \longrightarrow b + c$; 2 a species are removed, and 1 new b and c species are created.

A *configuration* $\vec{C} \in \mathbb{N}^\Lambda$ of a CRN assigns integer counts to every species $\lambda \in \Lambda$, and we use notation $\vec{C}[\lambda]$ to denote that count. For a species $\lambda \in \Lambda$, we denote the configuration consisting of a single copy of λ and no other species as $\vec{\lambda}$. It is often useful to reference the set of species whose counts are not zero in a given configuration. In such cases, the notation $\{\vec{C}\}$ is used. Formally, $\{\vec{C}\} = \{\lambda \in \Lambda \mid \vec{C}[\lambda] > 0\}$, and when convenient and clear from the context, we further use $\{\vec{C}\}$ to denote the configuration (vector) representation in which each element has a single copy. Finally, let $|\vec{C}| = \sum_{\lambda \in \Lambda} C[\lambda]$ denote the total number of copies of all species in a configuration, sometimes referred to as the *volume* of $\vec{C}$.

A reaction $(\vec{R}, \vec{P})$ is said to be *applicable* in configuration $\vec{C}$ if $\vec{R} \leq \vec{C}$; in other words, a reaction is applicable if $\vec{C}$ has at least as many copies of each species as $\vec{R}$. If the reaction $(\vec{R}, \vec{P})$ is applicable, it results in configuration $\vec{C'} = \vec{C} - \vec{R} + \vec{P}$ if it occurs, and we write $\vec{C} \rightarrow^{(\Lambda,\Gamma)}_{crn} \vec{C'}$, or simply $\vec{C} \rightarrow \vec{C'}$ when the model and CRN are clear from context. We use the same notation as configuration vectors to denote the size ($|\vec{R}|$ and $|\vec{P}|$) and explicit content ($\{\vec{R}\}$ and $\{\vec{P}\}$) of reactants and products for a reaction.

▶ **Definition 1** (Discrete Chemical Reaction Network). *A discrete chemical reaction network (CRN) is an ordered pair* (Λ, Γ) *where* Λ *is an ordered alphabet of species, and* Γ *is a set of rules over* Λ.

▶ **Definition 2** (Basic CRN Dynamics). *For a CRN* (Λ, Γ) *and configurations* $\vec{A}$ *and* $\vec{B}$*, we say that* $\vec{A} \rightarrow^{(\Lambda,\Gamma)}_{crn} \vec{B}$ *if there exists a rule* $(\vec{R}, \vec{P}) \in \Gamma$ *such that* $\vec{R} \leq \vec{A}$*, and* $\vec{A} - \vec{R} + \vec{P} = \vec{B}$.

If there exists a finite sequence of configurations such that $\vec{C} \rightarrow \vec{C}_1 \rightarrow \ldots \rightarrow \vec{C}_n \rightarrow \vec{D}$, then we say that $\vec{D}$ is *reachable* from $\vec{C}$ and we write $\vec{C} \rightsquigarrow \vec{D}$. A configuration is said to be *terminal* if no reactions are applicable. We also define an *initial configuration* for a CRN as

its starting configuration. A *CRN System* T is then defined as a pair of a CRN model and its initial configuration. The following sections define extensions of the basic CRN model by way of defining modified dynamics.

2.1.2 Void Genesis CRNs

A *Void Genesis CRN* $\mathcal{C}_{\mathcal{VG}} = ((\Lambda, \Gamma), z)$ is a basic CRN with a *zero* species $z \in \Lambda$ whose count is incremented whenever the count of any species other than z goes to zero. See Figure 2d for an example.

▶ **Definition 3** (Void-Genesis Dynamics)**.** *For a Void-Genesis CRN* $((\Lambda, \Gamma), z \in \Lambda)$ *and configurations* $\overrightarrow{A}$, $\overrightarrow{B_t}$ *and* $\overrightarrow{B}$, *we say that* $\overrightarrow{A} \rightarrow^{(\Lambda,\Gamma)}_{\mathcal{C}_{\mathcal{VG}}} \overrightarrow{B}$ *if there exists a rule* $(\overrightarrow{R}, \overrightarrow{P}) \in \Gamma$ *such that* $\overrightarrow{R} \leq \overrightarrow{A}$, $\overrightarrow{A} - \overrightarrow{R} + \overrightarrow{P} = \overrightarrow{B_t}$, *and* $\overrightarrow{B} = \overrightarrow{B_t} + n \cdot \vec{z}$ *where* n *is the cardinality of* $\{\lambda \in \Lambda \setminus \{z\} \mid \overrightarrow{A}[\lambda] \neq 0$ *and* $\overrightarrow{B_t}[\lambda] = 0\}$.

It is straightforward to show that the Void Genesis model is Turing-universal via simulating a register machine, and we do so in Section 4.7.

2.1.3 Inhibitory CRNs

A reaction γ is said to be *inhibited* by a species λ when the reaction γ may only be applied if λ is absent in the system. We define an inhibitor mapping $\mathcal{I} : \Gamma \rightarrow \mathbb{P}(\Lambda)$ that maps a reaction to a subset of species that inhibit the reaction. An *Inhibitory CRN* $\mathcal{C}_{\mathcal{IC}} = ((\Lambda, \Gamma), \mathcal{I})$ as defined by [7] is then a basic CRN along with the mapping $\mathcal{I}$. See Figure 2c for an example.

▶ **Definition 4** (Inhibitory Dynamics)**.** *For a Inhibitory CRN* $((\Lambda, \Gamma), \mathcal{I})$ *and configurations* $\overrightarrow{A}$ *and* $\overrightarrow{B}$, *we say that* $\overrightarrow{A} \rightarrow^{(\Lambda,\Gamma)}_{\mathcal{C}_{\mathcal{IC}}} \overrightarrow{B}$ *if there exists a rule* $\gamma = (\overrightarrow{R}, \overrightarrow{P}) \in \Gamma$ *such that* $\overrightarrow{R} \leq \overrightarrow{A}$, $\overrightarrow{A} - \overrightarrow{R} + \overrightarrow{P} = \overrightarrow{B}$, *and* $A[\lambda] = 0, \forall \lambda \in \mathcal{I}(\gamma)$.

2.1.4 Coarse-Rate CRNs

A *Coarse-Rate CRN* $\mathcal{C}_{CR} = ((\Lambda, \Gamma), rank)$ as introduced by [25] is a basic CRN along with a function $rank : \Gamma \rightarrow \mathbb{N}$. We define a set of reactions Γ^l as the set of all reactions γ where $rank(\gamma) = l$. The set of reactions Γ is then defined as an ordered partition set given by $\{\Gamma^1, \Gamma^2, \ldots, \Gamma^n\}$. Any applicable reaction γ^ℓ_i may only be applied if no reaction $\gamma^k_j \in \Gamma^k, \forall k \in [\ell + 1, n]$ is applicable. We use $(\overrightarrow{R}, \overrightarrow{P})^\ell$ to denote a reaction $\gamma^\ell \in \Gamma^\ell$. In the context of this paper we focus on models with rank at most 2. For clarity, we will refer to reactions with rank 2 as *fast* reactions, and the ones with rank 1 as *slow* reactions. See Figure 2b for an example.

▶ **Definition 5** (Coarse-Rate Dynamics)**.** *For a Coarse-Rate CRN* $((\Lambda, \Gamma), rank)$ *and configurations* $\overrightarrow{A}$ *and* $\overrightarrow{B}$, *we say that* $\overrightarrow{A} \rightarrow^{(\Lambda,\Gamma)}_{\mathcal{C}_{\mathcal{CR}}} \overrightarrow{B}$ *if there exists a rule* $(\overrightarrow{R}, \overrightarrow{P})^\ell \in \Gamma$ *such that* $\overrightarrow{R} \leq \overrightarrow{A}$, $\overrightarrow{A} - \overrightarrow{R} + \overrightarrow{P} = \overrightarrow{B}$, *and* $\nexists$ *an applicable reaction* $\gamma^{k>\ell}$.

2.1.5 Step-Cycle CRNs

A *step-cycle CRN* is a step CRN [2] that infinitely repeats steps 0 through $k-1$: that is, once $\overrightarrow{S}_{k-1}$ is added to the terminal configuration in the $(k-1)^{th}$ step, the resulting configuration is treated as the new initial configuration for the step CRN. More formally, a *step-cycle CRN* of k steps is an ordered pair $((\Lambda, \Gamma), (\overrightarrow{S}_0, \overrightarrow{S}_1, \ldots, \overrightarrow{S}_{k-1}))$, where the first element is a normal CRN (Λ, Γ) and the second is a sequence of length-$|\Lambda|$ vectors of non-negative

integers denoting how many copies of each species type to add after each step. We define a *step-configuration* $\overrightarrow{C}_i$ for a step-cycle CRN as a valid configuration $\overrightarrow{C}$ over (Λ, Γ) along with an integer $i \in \{0, \ldots, k-1\}$ that denotes the configuration's step.

▶ **Definition 6** (Step-Cycle Dynamics). *For a step-cycle CRN* $((\Lambda, \Gamma), (\overrightarrow{S}_0, \overrightarrow{S}_1, \ldots, \overrightarrow{S}_{k-1}))$ *and step-configurations* $\overrightarrow{A}_i$ *and* $\overrightarrow{B}_j$*, we say that* $\overrightarrow{A}_i \to_{\mathcal{C}_{SC}}^{(\Lambda,\Gamma)} \overrightarrow{B}_j$ *if either*

1. $i = j$*, there exists a rule* $(\overrightarrow{R}, \overrightarrow{P}) \in \Gamma$ *s.t.* $\overrightarrow{R} \leq \overrightarrow{A}_i$*, and* $\overrightarrow{A}_i - \overrightarrow{R} + \overrightarrow{P} = \overrightarrow{B}_j$*, or*
2. $(i+1) \mod k = j$*,* $\overrightarrow{A}_i$ *is terminal, and* $\overrightarrow{A}_i + \overrightarrow{S}_i = \overrightarrow{B}_j$*.*

2.1.6 Unique Instruction Parallel CRNs

The Unique Instruction Parallel model modifies the dynamics of a normal CRN (Λ, Γ) by applying a maximal set of compatible rules as a single transition. In this paper, we restrict this maximal set to contain only one application of any given rule, leaving the study of more relaxed parallel models for future work.

▶ **Definition 7** (Plausibly Parallel Rules). *A multiset of* n *(not necessarily distinct) rules* $\{(\overrightarrow{R}_1, \overrightarrow{P}_1), \ldots, (\overrightarrow{R}_n, \overrightarrow{P}_n)\}$ *are plausibly parallel for a configuration* $\overrightarrow{C}$ *over* Λ *if the vector* $\overrightarrow{R} = \sum_{i=1}^{n} \overrightarrow{R}_i$ *is such that* $\overrightarrow{R} \leq \overrightarrow{C}$*.*

▶ **Definition 8** (Unique-Instruction Plausibly Parallel). *A plausibly parallel multiset is said to be Unique-Instruction if it contains at most one copy of any given rule (i.e., it is a set). It is considered unique-instruction maximal if it is not a proper subset of any other unique-instruction plausibly parallel set.*

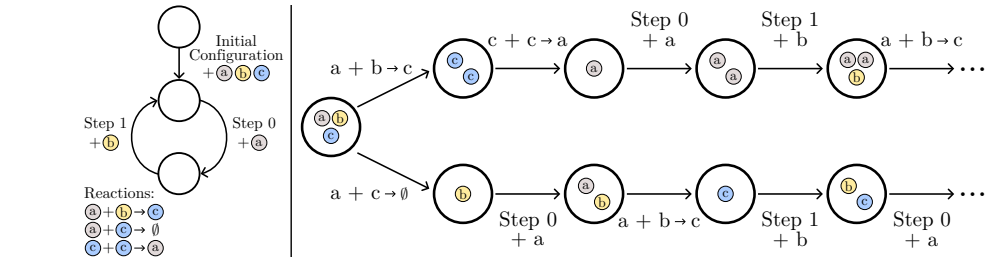
▶ **Definition 9** (Unique-Instruction Parallel Dynamics). *For a CRN* (Λ, Γ) *and configurations* $\overrightarrow{A}$ *and* $\overrightarrow{B}$*, we say that* $\overrightarrow{A} \to_{\mathcal{C}_{UI}}^{(\Lambda,\Gamma)} \overrightarrow{B}$ *if there exists a unique-instruction maximal plausibly parallel set* $\{(\overrightarrow{R}_1, \overrightarrow{P}_1), \ldots, (\overrightarrow{R}_k, \overrightarrow{P}_k)\}$ *for configuration* $\overrightarrow{A}$ *and rule set* Γ *such that* $\overrightarrow{B} = \overrightarrow{A} - \sum_{i=1}^{k} \overrightarrow{R}_i + \sum_{i=1}^{k} \overrightarrow{P}_i$*.*

2.2 Simulation

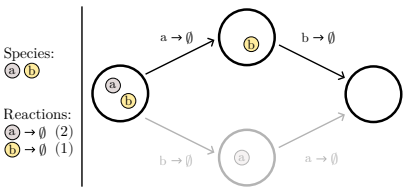
By way of Petri nets, discrete CRNs have seen various model extensions. To meaningfully compare the computational capabilities of these variants, we turn to the notion of simulation, which serves as a tool to compare the relative expressive power of each model. However, existing definitions in the literature vary in scope and applicability. Some emphasize strict structural correspondence while others focus purely on dynamic behavior. Thus, it is worthwhile to discuss why we formulate our own definition of simulation and equivalence.

Borrowed from classical process theory, (strong) *bisimulation* [21, 12, 4] is perhaps the most stringent form of equivalence. It requires that for every state and transition in one system, there exists a matching state and transition in the other, and vice versa. This strong bidirectional constraint means bisimulation ensures both behavioral and structural fidelity, and notably also implicitly guarantees efficiency.

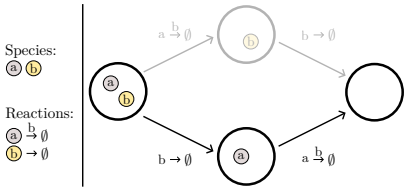
Weak bisimulation [10, 17], on the other hand, relaxes the strict step-by-step matching of bisimulation. Instead, a transition in one system may correspond to a *macrotransition* in the other: a sequence of transitions possibly allowing "hidden" or "silent" intermediate steps. This makes bisimulation more flexible and applicable to realistic implementations, but it is not without its own limitations. Pathway decomposition, as presented in [26, 27], takes a different approach. Rather than comparing reactions directly, they identify a CRN's



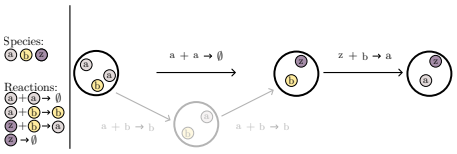
(a) Step-Cycle Example.



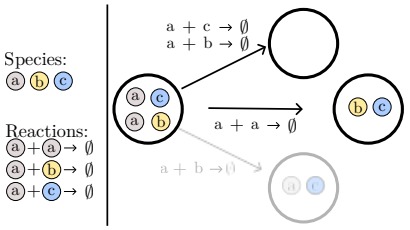
(b) Coarse-Rate Example.



(c) iCRN Example.



(d) Void Genesis Example.



(e) Unique Instruction Parallel Example.

Figure 2 Example systems for the 5 CRN models. The blurred portions of the figure represent invalid reaction sequences. (a) Step-cycle with 2 steps. Species are added when configurations reach a terminal state. (b) Coarse-rate. The numbers next to each reaction denote the rank. The bottom reaction sequence is invalid as the reaction with rank 2 must occur first. (c) Inhibitory. The top reaction sequence is invalid as there exists a b in the initial configuration, which is an inhibitor for the rule $a \to \emptyset$. (d) Void Genesis. Once the a species reaches a count of 0, the z species is created. In the bottom reaction sequence, the z species reacts with the b species, and another z species is created. (e) Unique-Instruction Parallel. The bottom reaction sequence is invalid as the rule $a + b \to \emptyset$ is not a maximal set.

formal basis – a set of "indivisible" formal pathways that collectively define its behavior. This allows pathway decomposition to capture phenomena such as *delayed choice*, in which nondeterministic behavior is distributed across multiple steps.

Our framework seeks to strike a middle ground. We define simulation in terms of a configuration map and macrotransitions, retaining the core idea of weak bisimulation but allowing for a more general structural correspondence between systems. Since this correspondence is so general, we explicitly include a measure of efficiency with our definition. We define polynomial efficient simulation that permits abstraction and internal nondeterminism, like weak bisimulation and pathway decomposition, but captures both dynamic behavior and bounded structural transformation. We say two systems are polynomially equivalent if they can each simulate each other via our definition. This is analogous to bisimilarity [21], but in the context of efficient, weak simulation.

Simulation Definition. To define the concept of one CRN system T' simulating another CRN system T we introduce *configuration maps* and *representative configurations*. A configuration map is a polynomial-time computable function $M : \text{configs}_{T'} \to \text{configs}_T \bigcup \{\perp\}$

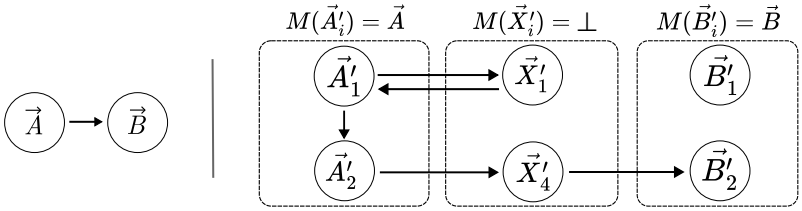


Figure 3 (Left) A system T with states $\overrightarrow{A}$ and $\overrightarrow{B}$, and transition $\overrightarrow{A} \to_T \overrightarrow{B}$. (Right) A state-space diagram for system T' that simulates T. Here, each arrow represents some transition $\overrightarrow{X} \to_{T'} \overrightarrow{Y}$ under the dynamics of T'. Observe how T follows T' and T' models T.

mapping at least one element in $\text{configs}_{T'}$ to each element in configs_T, and the *representative* configurations for a configuration $\overrightarrow{C} \in \text{configs}_T$ are $[\![\overrightarrow{C}]\!] = \{\overrightarrow{C'} \mid \overrightarrow{C} = M(\overrightarrow{C'})\}$, also computable in polynomial time.[1] Finally, we define the concept of *single-step transition*, $\overrightarrow{A} \to_T \overrightarrow{B}$, to mean that $\overrightarrow{A}$ transitions to $\overrightarrow{B}$ in system T. And, the concept of *macro transitionable*, $\overrightarrow{A'} \Rightarrow_{T'} \overrightarrow{B'}$, to mean that $\overrightarrow{B'}$ is reachable from $\overrightarrow{A'}$ in system T' through a sequence of k intermediate configurations $\langle \overrightarrow{A'}, \overrightarrow{X'_1}, \ldots, \overrightarrow{X'_k}, \overrightarrow{B'} \rangle$ such that $M(\overrightarrow{A'}) \neq \perp$, $M(\overrightarrow{B'}) \neq \perp$, and each $M(\overrightarrow{X'_i}) \in \{M(\overrightarrow{A'}), \perp\}$. Note that k can be zero, resulting in the sequence $\langle \overrightarrow{A'}, \overrightarrow{B'} \rangle$.

Intuition. Each representative configuration set $[\![\overrightarrow{C}]\!]$ is a particular collection (of at least 1 configuration) that adheres to the strictest modeling of system T within system T'. That is, any $\overrightarrow{C'} \in [\![\overrightarrow{C}]\!]$ must be able to grow into anything that $\overrightarrow{C} = M(\overrightarrow{C'})$ can grow into, and no $\overrightarrow{C'} \in [\![\overrightarrow{C}]\!]$ may grow into something that $\overrightarrow{C} = M(\overrightarrow{C'})$ cannot grow into.

For a given configuration map and collection of representative configurations, we define the concepts of *following* and *modeling*, followed by our definition of *simulation*.

▶ **Definition 10** (It Follows). *We say system T follows system T' if whenever $\overrightarrow{A'} \Rightarrow_{T'} \overrightarrow{B'}$ and $M(\overrightarrow{A'}) \neq M(\overrightarrow{B'})$, then $M(\overrightarrow{A'}) \to_T M(\overrightarrow{B'})$.*

▶ **Definition 11** (Models). *We say system T' models system T if $\overrightarrow{A} \to_T \overrightarrow{B}$ implies that $\forall \overrightarrow{A'} \in [\![\overrightarrow{A}]\!]$, $\exists \overrightarrow{B'} \in [\![\overrightarrow{B}]\!]$ such that $\overrightarrow{A'} \Rightarrow_{T'} \overrightarrow{B'}$.*

▶ **Definition 12** (Simulation). *A system T' simulates a system T if there exists a polynomial time computable function $M : \text{configs}_{T'} \to \text{configs}_T$ and polynomial time computable set $[\![\overrightarrow{C}]\!] = \{\overrightarrow{C'} \mid M(\overrightarrow{C'}) = \overrightarrow{C}\}$ for each $\overrightarrow{C} \in \text{configs}_T$, such that:*

1. *T follows T'.*
2. *T' models T.*

Polynomial Simulation. We say a simulation is *polynomial efficient* if the simulating system adheres to the following:

polynomial species and rules. The number of species and rules is at most polynomial in the number of species and rules of the simulated system.

polynomial rule size. The maximum rule size (number of products plus number of reactants) is polynomial in the maximum rule size of the simulated system.

[1] We write $M(\overrightarrow{C'}) = \perp$ to mean that the mapping is undefined, which is not the same as $M(\overrightarrow{C'}) = \overrightarrow{0}$.

polynomial transition sequences. For all B such that $A \to_T B$, the expected number of transitions taken to perform a macro transition from $M(A) \Rightarrow_{T'} M(B)$, conditioned that $M(A)$ does macro transition to $M(B)$, has expected number of transitions polynomial in the number of rules and species of the simulated system based on a uniform sampling of applicable rules.

polynomial volume. Each $\overrightarrow{C'} \in [\![\overrightarrow{C}]\!]$ has a volume that is polynomially bounded by the volume of C, and for any macro transition $A' \Rightarrow_{T'} B'$, any intermediate configuration within this macrotransition has volume polynomially bounded in the volume of $M(A')$ and $M(B')$.

▶ **Theorem 13** (Transitivity). *Given three CRN systems T_1, T_2 and T_3 such that T_2 simulates T_1 under polynomial simulation, and T_3 simulates T_2 under polynomial simulation, then T_3 simulates T_1 under polynomial simulation.*

Proof. Given three CRN systems T_1, T_2 and T_3 such that T_2 simulates T_1 under polynomial simulation, and T_3 simulates T_2 under polynomial simulation. Let $M_{21} : \text{configs}_{T_2} \to \text{configs}_{T_1}$ and $M_{32} : \text{configs}_{T_3} \to \text{configs}_{T_2}$ be the polynomial-time computable configuration mappings. We define a configuration mapping function $M_{31} : \text{configs}_{T_3} \to \text{configs}_{T_1}$ by composing functions M_{21} and M_{32} as follows.

$$M_{31}(\overrightarrow{C_3}) = \begin{cases} (M_{21} \circ M_{32})(\overrightarrow{C_3}) & \text{if } M_{32}(\overrightarrow{C_3}) \neq \perp \wedge M_{21}(M_{32}(\overrightarrow{C_3})) \neq \perp \\ \perp & \text{otherwise} \end{cases}$$

And the set of representative configurations for a configuration $\overrightarrow{C_1} \in \text{configs}_{T_1}$ as $[\![\overrightarrow{C_1}]\!] = \{\overrightarrow{C_3} \mid \overrightarrow{C_1} = M_{31}(\overrightarrow{C_3})\}$. If the configuration $\overrightarrow{C_1}$ has a non-empty set of representative configurations in T_2, and each of those configurations have representative configurations in T_3, then this set will contain all such configurations. Therefore, if both M_{21} and subsequently M_{32} are defined, this set will be non-empty.

We now show that T_3 simulates T_1 by proving that T_1 follows T_3 and T_3 models T_1 for the configuration mapping and representative configurations defined above. We then show that the simulation is polynomial efficient.

***T_1* follows *T_3*.** T_1 follows T_3 if for any two configurations $\overrightarrow{A_3}$ and $\overrightarrow{B_3}$ in T_3 where $M_{31}(\overrightarrow{A_3})$ and $M_{31}(\overrightarrow{B_3})$ are defined, such that $\overrightarrow{A_3} \Rightarrow_{T_3} \overrightarrow{B_3}$, and $M_{31}(\overrightarrow{A_3}) \neq M_{31}(\overrightarrow{B_3})$, then $M_{31}(\overrightarrow{A_3}) \to_{T_1} M_{31}(\overrightarrow{B_3})$. For these configurations if $M_{32}(\overrightarrow{A_3}) \neq M_{32}(\overrightarrow{B_3})$ then $M_{32}(\overrightarrow{A_3}) \to_{T_2} M_{32}(\overrightarrow{B_3})$. This is true because T_2 follows T_3.
Because T_1 follows T_2, for any two configurations $\overrightarrow{A_2}$ and $\overrightarrow{B_2}$ in T_2, if $\overrightarrow{A_2} \Rightarrow_{T_2} \overrightarrow{B_2}$, and $M_{21}(\overrightarrow{A_2}) \neq M_{21}(\overrightarrow{B_2})$, then $M_{21}(\overrightarrow{A_2}) \to_{T_1} M_{21}(\overrightarrow{B_2})$. A single-step transition is simply a macro-transition with one step. Therefore, if $M_{21}(M_{32}(\overrightarrow{A_3})) \neq M_{21}(M_{32}(\overrightarrow{B_3}))$, then $M_{21}(M_{32}(\overrightarrow{A_3})) \to_{T_1} M_{21}(M_{32}(\overrightarrow{B_2}))$. When M_{31} is defined we can infer based on the definition of M_{31} that, if $M_{31}(\overrightarrow{A_3}) \neq M_{31}(\overrightarrow{B_3})$ then $M_{31}(\overrightarrow{A_3}) \to_{T_1} M_{31}(\overrightarrow{B_3})$. Therefore, T_1 follows T_3.

***T_3* models *T_1*.** T_3 models T_1 if for any two configurations $\overrightarrow{A_1}$ and $\overrightarrow{B_1}$ in T_1 such that $\overrightarrow{A_1} \to_{T_1} \overrightarrow{B_1}$, $\forall \overrightarrow{A_3} \in [\![\overrightarrow{A_1}]\!]$, $\exists \overrightarrow{B_3} \in [\![\overrightarrow{B_1}]\!]$ under M_{31} such that $\overrightarrow{A_3} \Rightarrow_{T_3} \overrightarrow{B_3}$. For all such configurations $\overrightarrow{A_1}$ and $\overrightarrow{B_1}$ in T_1, $\forall \overrightarrow{A_2} \in [\![\overrightarrow{A_1}]\!], \exists \overrightarrow{B_2} \in [\![\overrightarrow{B_1}]\!]$ under M_{21} such that $\overrightarrow{A_2} \Rightarrow_{T_2} \overrightarrow{B_2}$ because T_2 models T_1. This macro transition $\overrightarrow{A_2} \Rightarrow_{T_2} \overrightarrow{B_2}$ is represented as a sequence of single-step transitions $\overrightarrow{X_2}^i \to_{T_2} \overrightarrow{X_2}^{i+1}$ where $M_{21}(\overrightarrow{X_2}^i), M_{21}(\overrightarrow{X_2}^{i+1}) \in \{\overrightarrow{A_1}, \perp\}$ for $0 \leq i < k$ as shown in Figure 4. Furthermore, $\forall \overrightarrow{X_3^j} \in [\![\overrightarrow{X_2}^i]\!]$, $\exists \overrightarrow{X_3}^\ell \in [\![\overrightarrow{X_2}^{i+1}]\!]$ under M_{32} such that $\overrightarrow{X_3}^j \Rightarrow_{T_3} \overrightarrow{X_3}^\ell$ because T_3 models T_2.

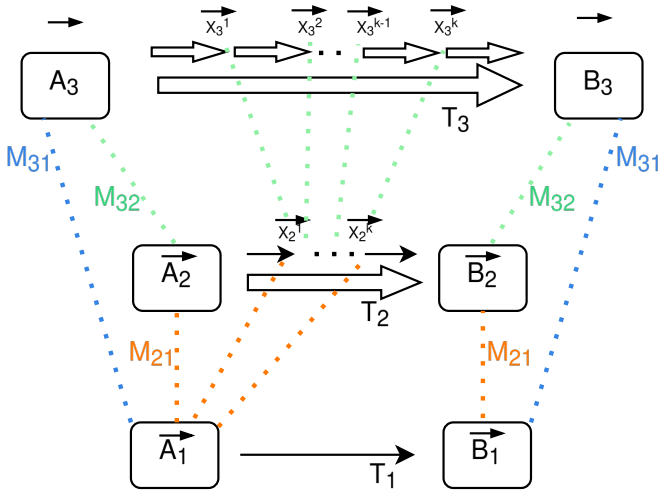


Figure 4 A single-step transition $\overrightarrow{A_1} \to_{T_1} \overrightarrow{B_1}$ is simulated using a sequence of transitions starting at $\overrightarrow{A_2}$ through $\overrightarrow{B_2}$ in T_2. Each of these single-step is represented using a sequence of transitions in T_3.

Because $M_{32}(\overrightarrow{X_3}^j) = \overrightarrow{X_2}^i$ and $M_{32}(\overrightarrow{X_3}^\ell) = \overrightarrow{X_2}^{i+1}$ and $M_{21}(\overrightarrow{X_2}^i)$, $M_{21}(\overrightarrow{X_2}^{i+1}) \in \{\overrightarrow{A_1}, \perp\}$, therefore, $M_{31}(\overrightarrow{X_3}^j)$, $M_{31}(\overrightarrow{X_3}^\ell) \in \{\overrightarrow{A_1}, \perp\}$ for $0 \leq j < k$ and $j < \ell \leq k$. As shown in Figure 4, the sequence of macro-transitions $\overrightarrow{X_3}^j \Rightarrow_{T_3} \overrightarrow{X_3}^\ell$ can be represented as a single macro-transition $\langle \overrightarrow{A_3}, \overrightarrow{X_3}^1, \ldots, \overrightarrow{X_3}^k, \overrightarrow{B_3} \rangle$, where $M_{31}(\overrightarrow{X_3}^j) \in \{\overrightarrow{A_1}, \perp\}$ for $0 \leq j \leq k$. Also, $M_{32}(\overrightarrow{A_3}) = \overrightarrow{A_2}$ and $M_{32}(\overrightarrow{B_3}) = \overrightarrow{B_2}$. We know that $M_{21}(\overrightarrow{A_2}) = \overrightarrow{A_1}$ and $M_{21}(\overrightarrow{B_2}) = \overrightarrow{B_1}$, therefore, $M_{31}(\overrightarrow{A_3}) = \overrightarrow{A_1}$ and $M_{31}(\overrightarrow{B_3}) = \overrightarrow{B_1}$ when both M_{21} and M_{32} are defined. Hence, such a macro-transition $\overrightarrow{A_3} \Rightarrow_{T_3} \overrightarrow{B_3}$ models the single-step transition $\overrightarrow{A_1} \to_{T_1} \overrightarrow{B_1}$ when $\overrightarrow{A_3} \in [\![\overrightarrow{A_1}]\!]$ and $\overrightarrow{B_3} \in [\![\overrightarrow{B_1}]\!]$. Therefore, T_3 models T_1.

Polynomial Simulation. Given that T_2 simulates T_1 under polynomial simulation, and T_3 simulates T_2 under polynomial simulation, we now show that T_3 simulates T_1 under polynomial simulation. Let Λ_i and Γ_i be the set of species and reactions for the model T_i for $1 \leq i \leq 3$.

1. **polynomial species and rules:** We know that $|\Lambda_2| = \mathcal{O}(|\Lambda_1|) = |\Lambda_1|^{c_1}$ and $|\Lambda_3| = \mathcal{O}(|\Lambda_2|) = |\Lambda_2|^{c_2}$, therefore, $|\Lambda_3| = |\Lambda_1|^{c_1 \cdot c_2}$. Similarly, $|\Gamma_3| = |\Gamma_1|^{c_1 \cdot c_2}$. Therefore, $|\Lambda_3| = \mathcal{O}(|\Lambda_1|)$ and $|\Gamma_3| = \mathcal{O}(|\Gamma_1|)$.
2. **polynomial rule size:** Let $\mathcal{R}_i$ be the largest rule in the model T_i. We know that $|\mathcal{R}_2| = \mathcal{O}(|\mathcal{R}_1|)$ and $|\mathcal{R}_3| = \mathcal{O}(|\mathcal{R}_2|)$, therefore, $|\mathcal{R}_3| = \mathcal{O}(|\mathcal{R}_1|)$.
3. **polynomial transition sequences:** The length of the transition sequence for a macro-transition in T_3 is bounded by $\mathcal{O}(|\Lambda_2| + |\Gamma_2|)$. Given that $|\Lambda_2| = |\Lambda_1|^{c_1}$ and $|\Gamma_2| = |\Gamma_1|^{c_1}$, therefore, each macro-transition in T_3 simulating a single-step in T_1 uses $\mathcal{O}(|\Lambda_1| + |\Gamma_1|)$ single-step transitions.
4. **polynomial volume:** For a macro-transition $\overrightarrow{A_2} \Rightarrow_{T_2} \overrightarrow{B_2}$ representing a transition $\overrightarrow{A_1} \to_{T_1} \overrightarrow{B_1}$, the volume of representative configurations $\overrightarrow{A_2}$ and $\overrightarrow{B_2}$ is polynomial in the volume of configurations $\overrightarrow{A_1}$ and $\overrightarrow{B_1}$ respectively. Similarly, for a macro-transition $\overrightarrow{A_3} \Rightarrow_{T_3} \overrightarrow{B_3}$ as shown in Figure 4, the volume of $\overrightarrow{A_3}$ and $\overrightarrow{B_3}$ is polynomial in volume of $\overrightarrow{A_2}$ and $\overrightarrow{B_2}$ respectively. Therefore, the volume of representative configurations $\overrightarrow{A_3}$ and $\overrightarrow{B_3}$ is polynomial in the volume of configurations $\overrightarrow{A_1}$ and $\overrightarrow{B_1}$, where $\overrightarrow{A_3} \Rightarrow_{T_3} \overrightarrow{B_3}$ models $\overrightarrow{A_1} \to_{T_1} \overrightarrow{B_1}$. The volume of all intermediate configurations are bound by $\overrightarrow{A_3}$ and $\overrightarrow{B_3}$. ◀

3 Detecting Zero: Examples in Each Model

The power each extended CRN model has over the basic CRN model stems from the ability to detect when a species has reached a count of zero. Thus, the focus of this section is twofold: to detail how each model is capable of detecting zero and to provide minimum working examples (MWEs) for ease of comprehension. For all examples, we want to check if species s is in the system. The idea is that there is a species c_s that turns to n_s if there are no s's and y_s if there are s's in the system while nothing else in the system changes.

Void Genesis. Detecting zero is explicitly done by the model. For the simplest version of the model, we assume we have no z species. To check whether s is zero, simply use either as a catalyst (examples on the left side). For this to work, you must maintain a single z. Thus, when producing any species s, continue to check with s as a catalyst and z as a reactant. As an example, the rule $x \to s$ would be modified to be implemented with the two rules on the right side.

$c_s + s \to y_s + s$ $\qquad$ $x + s \to 2s$

$s_s + z \to n_s + z$ $\qquad$ $x + z \to s$

Inhibitory CRNs. We can use an inhibited rule and a catalyst to detect whether a species s exists or not.

$c_s \overset{s}{\to} n_s$ runs when s does not exist

$c_s + s \to y_s + s$ runs when s does exist

Course-Rate CRNs. Detecting a species s only requires that there is a fast rule that uses it and a slow rule that does not.

Fast Reaction $c_s + s \to y_s + s$ (will always execute first)

Slow Reaction $c_s \to n_s$ (will only execute if no s exists in the system)

Step-Cycle CRNs. Detecting zero is simple by using s in a reaction and then going to another step. Since each step reaches a terminal configuration, it must not exist if the reaction did not occur.

Step	Description	Add	Rules
1	Add something that only reacts with s	a single c_s	$s + c_s \to y$
2	Check if it reacted	a single w	$w + y \to y_s + s$ (s exists) $w + c_s \to n_s$ (s not exists)

Unique-Instruction Parallel CRNs. The parallel model UI takes advantage of how rules are applied to force reactions to run in a specific order depending on whether the count of a certain species is zero or not. Since we are limiting the possible rules that can run by our species selection, detecting zero can be done in this manner. The intuition is to run two independent rules, followed by a rule that uses the output of both. The "Round" indicates that all rules in this round would execute before the next round due to the maximal selection.

Round	Description	Rules
1	Create a timing/clock species (t_i) and a species to use s with.	$c_s \to r_c + t_1$
2	Try to use s and use the timer t in another rule.	$r_c + s \to r_s$ (can only run if s exists) $t_1 \to t_2$ (runs even if the other rule can not)
3	Now we use the timer to see if the other rule ran	$t_2 + r_s \to y_s + s$ (there is an s in the system) $t_2 + r_c \to n_s$ (there are no s's in the system)

4 Equivalence

This section shows equivalence between the CRN models, represented as the purple bi-directional arrows in Figure 1. We first introduce a more general Void Genesis model, along with some useful notation and techniques, before presenting our equivalence results. A full version of the paper, including construction details and formal proofs, is available on arXiv.

4.1 Equivalence Preliminaries

4.1.1 k-Void Genesis

Due to the complexity of some of the simulations, we first provide a more general version of the VG model that makes simulation easier. We will prove that the standard Void Genesis model can still simulate the more general model with at most a polynomial blow-up in rules, species, and expected rule applications. Essentially, the only difference is that rather than a single zero species z, there can be a different zero species for every species.

Formally, a *k-Void Genesis CRN* $\mathcal{C}_{\mathcal{KVG}} = ((\Lambda, \Gamma), Z_\emptyset)$ is a CRN with a partial mapping function $Z_\emptyset : \Lambda_1 \to \Lambda_2$, such that $\Lambda_1 \cup \Lambda_2 = \Lambda$ and $\Lambda_1 \cap \Lambda_2 = \emptyset$, that indicates which species is created whenever another species count goes to zero (if mapped). The partition creates a distinction between *normal* species and the special *zero-counting* species, which eliminates chaining effects that could arise with zero-species being created from the elimination of other zero-species. For convenience, we use the notation $Z_\emptyset(\lambda) \to z_\lambda$ to indicate that if the count of species λ in the system goes to zero, a z_λ species is created.

▶ **Definition 14** (k-Void-Genesis Dynamics). *For a k-Void-Genesis CRN $((\Lambda, \Gamma), Z_\emptyset)$ and configurations $\overrightarrow{A}$, $\overrightarrow{B_t}$ and $\overrightarrow{B}$, we say that $\overrightarrow{A} \to^{(\Lambda,\Gamma)}_{\mathcal{C}_{\mathcal{KVG}}} \overrightarrow{B}$ if there exists a rule $(\overrightarrow{R}, \overrightarrow{P}) \in \Gamma$ such that $\overrightarrow{R} \leq \overrightarrow{A}$, $\overrightarrow{A} - \overrightarrow{R} + \overrightarrow{P} = \overrightarrow{B_t}$, and $\overrightarrow{B} = \overrightarrow{B_t} + \overrightarrow{Z}$ where $\overrightarrow{Z} = \sum_{\lambda \in C} \vec{z}_\lambda$, and $C = \{\lambda \in \Lambda | \overrightarrow{A}[\lambda] \neq 0$ and $\overrightarrow{B_t}[\lambda] = 0\}$.*

4.1.2 Notation and Techniques

Notation. To simplify the proofs and for consistency, we use the following notation for rules. For rule i, we have $\mathcal{G}_i = (\overrightarrow{\mathcal{R}_i}, \overrightarrow{\mathcal{P}_i})$. We make use of the $\{\mathcal{R}\}$ and $|\mathcal{R}|$ notation defined in the preliminaries, as well the difference between a configuration with many species $\overrightarrow{X}$ versus a configuration with only a single species $\vec{X}$ by the over arrow used.

One technique that is used in our simulations is to maintain a single global leader species G that selects which rule $\mathcal{G}_i$ to execute, and then a sequence of rules to either sequentially consume the reactants or sequentially break down the product of the other models' rules to execute zero-checking before managing zero species.

Sequential Reactants. Some of the results also benefit from processing the reactants of a rule one at a time as well. This is needed when consuming a species might result in its count as zero and some action needs to be taken. In order to verify that all reactants exists (and thus the rule could be executed), we use all reactants as a catalyst with a global leader species G. Table 1a describes the necessary rules for consuming reactants sequentially. For rule i, this creates R_i^1 that will begin the chain of consuming one species at a time. We create a C_i^j species in order to check whether r_j is now zero or not. This step can be skipped if no check is needed. We create the species R_i^j sequentially, which is then consumed if r_j is in the system. Systems that use this technique will have one copy of G in the initial configuration and no R or C species. For shorthand, we will represent this series of rules as $\vec{G} + \overrightarrow{\mathcal{R}_i} \dashrightarrow \vec{G} + \overrightarrow{\mathcal{P}_i}$.

Table 1 An overview of two techniques. (a) A procedure to consume the reactants of some rule sequentially in order to test whether a rule can be applied and whether zero species are (or need to be) created. This is abbreviated for rule i as $\overrightarrow{\mathcal{R}_i} \dashrightarrow \overrightarrow{\mathcal{P}_i}$. (b) Creating products sequentially by consuming all reactants and then using counter species to create each product with a different rule until all have been created. This may be needed to decrease the number of rule combinations while handling zero species. Note that the zero species z_{p_j} is consumed when p_j is created. If p_j still exists, then it is used as a catalyst. This process is denoted for some rule i as $\overrightarrow{\mathcal{R}_i} \twoheadrightarrow\twoheadrightarrow \overrightarrow{\mathcal{P}_i}$.

	$\vec{G} + \overrightarrow{\mathcal{R}_i} \to \vec{R}_i^1 + \overrightarrow{\mathcal{R}_i}$
$\forall r_j \in \mathcal{R}_i$:	$\vec{R}_i^j + \vec{r}_j \to \vec{C}_i^j$ (check for zero) $\vec{C}_i^j + \vec{z}_{r_j} \to \vec{R}_i^{j+1} + \vec{z}_{r_j}$ (r_j is zero) $\vec{C}_i^j + \vec{r}_j \to \vec{R}_i^{j+1} + \vec{r}_j$ (r_j is not zero)
	$\vec{R}_i^{\lvert\mathcal{R}_i\rvert+1} \to \vec{G} + \overrightarrow{\mathcal{P}_i}$

(a) Using reactants sequentially $\overrightarrow{\mathcal{R}_i} \dashrightarrow \overrightarrow{\mathcal{P}_i}$.

	$\vec{G} + \overrightarrow{\mathcal{R}_i} \to \vec{P}_i^1$
$\forall p_j \in \mathcal{P}_i$:	$\vec{z}_{p_j} + \vec{P}_i^j \to \vec{p}_j + \vec{P}_i^{j+1}$ (remove zero) $\vec{p}_j + \vec{P}_i^j \to \vec{p}_j + \vec{p}_j + \vec{P}_i^{j+1}$ (no zero)
	$\vec{P}_i^{\lvert\mathcal{P}_i\rvert+1} \to \vec{G}$

(b) Creating products sequentially $\overrightarrow{\mathcal{R}_i} \twoheadrightarrow\twoheadrightarrow \overrightarrow{\mathcal{P}_i}$.

Sequential Products. This technique only allows a single G species to exist in the system at a time. Table 1b describes the necessary rules where a rule $\mathcal{G}_i = (\overrightarrow{\mathcal{R}_i}, \overrightarrow{\mathcal{P}_i})$ is chosen if the reactants exist in the system. If this is the case, then each product is created sequentially by having the species $P_i^1, \ldots, P_i^{|\mathcal{P}_i|}$ in the system (one at a time, starting from P_i^1), then consuming each species P_i^j to create the product p_j and the next P_i^{j+1} until all products have been produced. If species P_i^j exists in the system, it means that for rule $\mathcal{G}_i$, the first $j-1$ products have been created, and either the next p_j will be created or the process will end (return G) if there are no more products.

In a simulating system, this requires an additional number of rules and species on the order of the largest number of products in any rule. The final rule returns the single leader species. Note that we make use of a zero species z_{p_j} for every product species p_j, i.e., $Z_\emptyset(p_j) = z_{p_j}$. For shorthand, we will represent this set of rules for rule i as $\overrightarrow{\mathcal{R}_i} \twoheadrightarrow\twoheadrightarrow \overrightarrow{\mathcal{P}_i}$. Thus, the entire set of rules and species from Table 1b, that initiate with a global leader G, would be given in shorthand as $\vec{G} + \overrightarrow{\mathcal{R}_i} \twoheadrightarrow\twoheadrightarrow \vec{G} + \overrightarrow{\mathcal{P}_i}$.

4.2 k-Void Genesis equivalence with Void Genesis

▶ **Lemma 15.** *The k-Void Genesis model can simulate the Void Genesis model.*

Construction. Given a Void Genesis model $\mathcal{C}_{\mathcal{VG}} = ((\Lambda, \Gamma), z)$, let $Z_\emptyset(\lambda) = z$, $\forall \lambda \in \Lambda \setminus \{z\}$. Then the k-Void Genesis model $\mathcal{C}_{\mathcal{KVG}} = ((\Lambda, \Gamma), Z_\emptyset)$ is equivalent.

▶ **Lemma 16.** *The Void Genesis model can simulate the k-Void Genesis model.*

Construction. This result is straightforward using the methods to sequentially consume reactants as previously defined. We slightly modify them with specifics as shown in Table 2. Whenever we consume a reactant r_j, we check if the single z species exists and if it does, create the specific z_{r_j} species if r_j is mapped in $Z_\emptyset$.

Given a k-Void Genesis model $\mathcal{C}_{\mathcal{KVG}} = ((\Lambda, \Gamma), Z_\emptyset)$, we create a VG CRN $\mathcal{C}_{\mathcal{VG}} = ((\Lambda', \Gamma'), z)$. We let $\Lambda' = \Lambda \cup \{G\} \cup \{z_\lambda : \lambda \in \Lambda \setminus \{z\}\} \cup S_R$ where $S_R = \{R_i^j : 1 \leq i \leq |\Gamma|, 1 \leq j \leq |\mathcal{R}_i| + 1\}$. We can simulate having specific z_j species for all j species by keeping the z species count at 0, and checking whether we consumed the last j.

Table 2 (a) The rules for the simulation of k-Void Genesis by Void Genesis. For each applicable rule, we create a set of rules that sequentially consume each reactant r_j, and if its count is now zero, it creates the z_{r_j} species (unless it is unmapped, then z is simply consumed). Only one of the zero check rules is added based on the mapping, which is why they are grouped. (b) The reactions that simulate the rule $\vec{a} + 2\vec{b} \to 2\vec{a} + \vec{c}$ from the k-VG model within the VG model.

$\vec{G} + \overrightarrow{\mathcal{R}_i} \to \vec{R}_i^1 + \overrightarrow{\mathcal{R}_i}$
$\vec{R}_i^j + \vec{r}_j \to \vec{C}_i^j$ (check for zero)
$\forall r_j \in \mathcal{R}_i : \begin{cases} \vec{C}_i^j + \vec{z} \to \vec{R}_i^{j+1} \text{(unmapped species)} \\ \vec{C}_i^j + \vec{z} \to \vec{R}_i^{j+1} + \vec{z}_{r_j} \text{(zero species)} \end{cases}$
$\vec{C}_i^j + \vec{r}_j \to \vec{R}_i^{j+1} + \vec{r}_j$ (count not zero)
$\vec{R}_i^{\lvert\mathcal{R}_i\rvert+1} \to \vec{G} + \overrightarrow{\mathcal{P}_i}$

(a) Zero checking reactants $\overrightarrow{\mathcal{R}_i}$ $\dashrightarrow$ $\overrightarrow{\mathcal{P}_i}$.

$\vec{G} + \vec{a} + 2\vec{b} \to \vec{R}_1^1 + \vec{a} + 2\vec{b}$	
$\vec{R}_1^1 + \vec{a} \to \vec{C}_1^1$	$\vec{C}_1^2 + \vec{b} \to \vec{R}_1^3 + \vec{b}$
$\vec{C}_1^1 + \vec{z} \to \vec{R}_1^2 + \vec{z}_a$	$\vec{R}_1^3 + \vec{b} \to \vec{C}_1^3$
$\vec{C}_1^1 + \vec{a} \to \vec{R}_1^2 + \vec{a}$	$\vec{C}_1^3 + \vec{z} \to \vec{R}_1^4 + \vec{z}_b$
$\vec{R}_1^2 + \vec{b} \to \vec{C}_1^2$	$\vec{C}_1^3 + \vec{b} \to \vec{R}_1^4 + \vec{b}$
$\vec{C}_1^2 + \vec{z} \to \vec{R}_1^3 + \vec{z}_b$	$\vec{R}_1^4 \to \vec{G} + 2\vec{a} + \vec{c}$

(b) Example reaction set.

Table 3 (a) Reactions for an Inhibitory CRN to simulate any given k-VG CRN. (b) Reactions for a k-VG CRN to simulate any given Inhibitory CRN.

$\forall \gamma_i \in \Gamma$:	1. $\overrightarrow{\mathcal{R}}_i \xrightarrow{I} \overrightarrow{e}_{\{\mathcal{R}_i\}} + \overrightarrow{\mathcal{P}}_i + \lvert\{\mathcal{R}_i\}\rvert \cdot \vec{I}$
$\forall \lambda_i \in \Lambda_1$:	2. $\vec{e}_{\lambda_i} + \vec{I} + \vec{\lambda}_i \to \vec{\lambda}_i$
	3. $\vec{e}_{\lambda_i} + \vec{I} \xrightarrow{\lambda_i} \vec{z}_{\lambda_i}$

(a) iCRN simulating k-VG.

$\forall \gamma_i \in \Gamma$:	1. $\vec{G} + \overrightarrow{\mathcal{R}_i} + \overrightarrow{Z_i} \to \vec{G}_i + \overrightarrow{\mathcal{R}_i} + \overrightarrow{Z_i}$
	2. $\vec{G}_i + \overrightarrow{\mathcal{R}_i} \twoheadrightarrow \vec{G} + \overrightarrow{\mathcal{P}_i}$

(b) k-VG simulating iCRN.

▶ **Theorem 17.** *The Void Genesis model is equivalent under polynomial simulation to the k-Void Genesis model.*

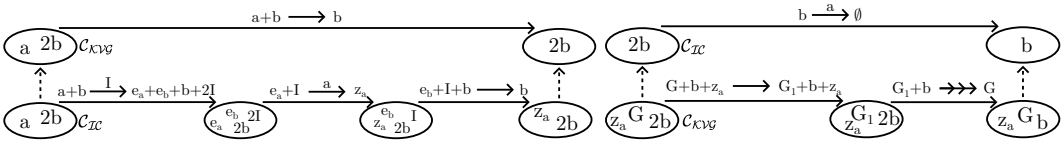
4.3 Inhibitory CRNs

▶ **Lemma 18.** *Inhibitory CRNs can simulate any given k-VG CRN under polynomial simulation.*

Construction. Given a k-VG CRN $\mathcal{C}_{\mathcal{KVG}} = ((\Lambda, \Gamma), Z_\emptyset)$, we construct an Inhibitory CRN $\mathcal{C}_{\mathcal{IC}} = ((\Lambda', \Gamma'), \mathcal{I})$. We let $\Lambda' = \Lambda \cup \{I, e_{\lambda_1}, \cdots, e_{\lambda_{|\Lambda_1|}}\}$ where I and $e_{\lambda_1}, \ldots, e_{\lambda_{|\Lambda_1|}}$ check if any species of Λ_1 used in a simulated reaction have a resulting count of zero. Recall that Λ_1 is the set of non-zero species in Λ. Each reaction in Γ γ_i is simulated using Reaction 1 from Table 3a. Reactions 2 and 3 check if any reactants of γ_i reached a count of zero following γ_i's simulation. If so, then a corresponding zero species is produced. Figure 5a shows an example of an Inhibitory CRN simulating a k-VG CRN with $\Lambda = \{a, b\}$ and a reaction that consumes the species a.

▶ **Lemma 19.** *k-VG CRNs can simulate any Inhibitory CRN under polynomial simulation.*

Construction. Given an Inhibitory CRN $\mathcal{C}_{\mathcal{IC}} = ((\Lambda, \Gamma), \mathcal{I})$, we construct a k-VG CRN $\mathcal{C}_{\mathcal{KVG}} = ((\Lambda', \Gamma'), Z_\emptyset)$. We let $\Lambda' = \Lambda \cup \{G, G_1, \ldots, G_{|\Gamma|}, P_i^1, \ldots, P_{|\Gamma|}^{|\mathcal{P}_{|\Gamma|}|+1}, z_{\lambda_1}, \ldots, z_{\lambda_{|\Lambda|}}\}$ where the global species G is consumed to produce G_i when selecting a reaction $\gamma_i \in \Gamma$. The z species are the zero-species, and the species $P_i^1 \ldots P_i^{|\mathcal{P}_i|+1}$ are produced for the sequential product generation as described in Table 1b. Let Z_i represent the set of z species corresponding to inhibitors of γ_i. Each reaction γ_i in Γ is represented by the two reactions given in Table 3b, where Reaction 1 checks if γ_i is applicable (indicated by the presence of reactants of γ_i and Z_i) and Reaction 2 applies the reaction by generating products sequentially.



(a) $\mathcal{C}_{\mathcal{IC}} \to \mathcal{C}_{\mathcal{KVG}}$ **(b)** $\mathcal{C}_{\mathcal{KVG}} \to \mathcal{C}_{\mathcal{IC}}$

Figure 5 (a) A rule application in the *simulated* k-void genesis CRN $\mathcal{C}_{\mathcal{KVG}}$ and the equivalent rule application sequence in the *simulating* inhibitory CRN $\mathcal{C}_{\mathcal{IC}}$. The dashed arrows represent mapping a configuration of $\mathcal{C}_{\mathcal{IC}}$ to a configuration of $\mathcal{C}_{\mathcal{KVG}}$. (b) The other direction.

Table 4 (a) Reactions for a Coarse-Rate CRN to simulate any given k-VG CRN. (b) Reactions for a k-VG CRN to simulate any given Coarse-Rate CRN.

Fast Reactions (Rank 2)	
$\forall\ \gamma_i \in \Gamma$:	1. $\vec{G} + \overrightarrow{\mathcal{R}_i} \to \vec{x} + \overrightarrow{e_{\{\mathcal{R}_i\}}} + \overrightarrow{\mathcal{P}_i}$
$\forall \lambda_i \in \Lambda_1$:	2. $\vec{x} + \vec{e}_{\lambda_i} + \vec{\lambda}_i \to \vec{x} + \vec{\lambda}_i$
	3. $\vec{y} + \vec{e}_{\lambda_i} \to \vec{y} + \vec{z}_{\lambda_i}$
Slow Reactions (Rank 1)	
	4. $\vec{x} \to \vec{y}$
	5. $\vec{y} \to \vec{G}$

(a) Coarse-Rate CRN simulating k-VG CRN.

	1. $\vec{G} + \vec{g}_i + \overrightarrow{\mathcal{R}_i} \to \vec{r}_1 + \overrightarrow{\mathcal{P}_i}$
$\forall \gamma_i^2 \in \Gamma^2$,	1b. $\vec{G} + \vec{g}_i + \vec{z}_k \to \vec{G} + \vec{z}_k$
$z_k \in \{\mathcal{R}_i\}$:	2. $\vec{r}_j + \vec{g}_j \to \vec{r}_{j+1} + \vec{g}_j$
	3. $\vec{r}_j + \vec{z}_{g_j} \to \vec{r}_{j+1} + \vec{g}_j$
	4. $\vec{r}_{\lvert\Gamma^2\rvert+1} \to \vec{G}$
	5. $\vec{G} + \vec{z}_{g_1} + \ldots + \vec{z}_{g_{\lvert\Gamma^2\rvert}} \to \vec{s}$
	6. $\vec{t} \to \vec{G} + \vec{g}_1 + \ldots + \vec{g}_{\lvert\Gamma^2\rvert}$
$\forall \gamma_i^1 \in \Gamma^1$:	7. $\vec{s} + \overrightarrow{\mathcal{R}_i} \to\to\to \vec{t} + \overrightarrow{\mathcal{P}_i}$

(b) k-VG CRN simulating Coarse-Rate CRN.

Figure 5b shows an example of a k-VG CRN simulating an Inhibitory CRN with $\Lambda = \{a, b\}$ and a reaction that consumes b if a is absent.

▶ **Theorem 20.** *The Void Genesis model is equivalent under polynomial simulation to the Inhibitory model.*

4.4 Coarse-Rate CRNs

▶ **Lemma 21.** *Coarse-Rate CRNs can simulate any given k-VG CRN under polynomial simulation.*

Construction. Given a k-VG CRN $\mathcal{C}_{\mathcal{KVG}} = ((\Lambda, \Gamma), Z_\emptyset)$, we construct a Course-Rate CRN $\mathcal{C}_{\mathcal{CR}} = ((\Lambda', \Gamma'), rank)$. Let $\Lambda' = \Lambda \cup \{x, y, G, e_{\lambda_1}, \cdots, e_{\lambda_{|\Lambda_1|}}\}$ where G is used to select a reaction and species x and y check if any non-zero species in Λ have a count of zero. Additionally, $e_{\lambda_1}, \cdots, e_{\lambda_{|\Lambda_1|}}$ are used to check the counts of the $\lambda_i \in \Lambda_1$ that are relevant to the simulated reaction, where Λ_1 is the set of non-zero species in Λ. As shown in Table 4a, each reaction $\gamma_i \in \Gamma$ is simulated using 3 fast reactions which check if any involved reactant reached a count of zero and generates the corresponding zero species if so, along with applying the original reaction. Finally the 2 slow reactions use up any x or y species in the system to produce G. This allows the system to apply another reaction in Γ. Figure 6 shows an example of a Coarse-Rate CRN simulating a k-VG CRN with $\Lambda = \{a, b\}$ and a reaction that consumes a, hence, producing z_a.

▶ **Lemma 22.** *k-VG CRNs can simulate any given Coarse-Rate CRN under polynomial simulation.*

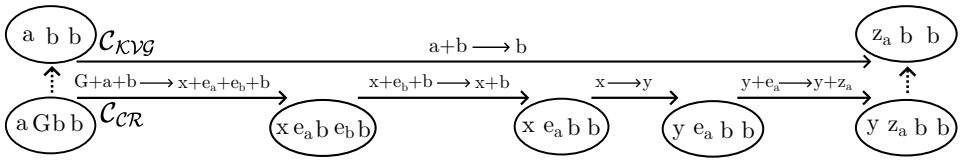


Figure 6 A rule application in the *simulated* k-void genesis CRN $\mathcal{C}_{\mathcal{KVG}}$ and the equivalent rule application sequence in the *simulating* Coarse-Rate CRN $\mathcal{C}_{\mathcal{CR}}$.

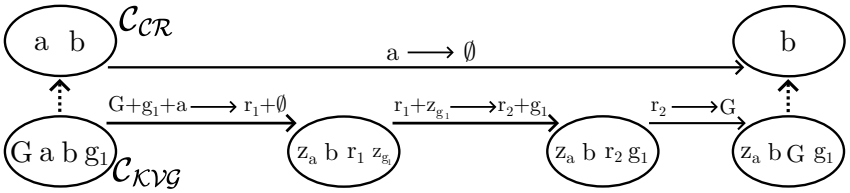


Figure 7 A rule application in the *simulated* Coarse-Rate CRN $\mathcal{C}_{\mathcal{CR}}$ and the equivalent rule application sequence in the *simulating* k-void genesis CRN $\mathcal{C}_{\mathcal{KVG}}$.

Construction. Given a Coarse-Rate CRN $\mathcal{C}_{\mathcal{CR}} = ((\Lambda, \Gamma), rank)$, we construct a k-VG CRN $\mathcal{C}_{\mathcal{KVG}} = ((\Lambda', \Gamma'), Z_\emptyset)$. Let $\Lambda' = \Lambda \cup \{G, g_1, \cdots, g_{|\Gamma^2|}, z_{g_1}, \cdots, z_{g_{|\Gamma^2|}}, z_{\lambda_1}, \cdots, z_{\lambda_{|\Lambda|}}, P_1^1, \cdots, P_{|\Gamma^1|}^{|\mathcal{P}_{|\Gamma^1|}|+1}, s, t\}$ as follows. G and a random g_i species is consumed to select a random fast reaction $\gamma_i^2 \in \Gamma^2$ as illustrated in Table 4*b*. We additionally add $P_1^1, \cdots, P_{|\Lambda|}^{|\mathcal{P}_{|\Lambda|}|+1}$ for generating the products of any slow reaction. We add species s to signify that no fast reactions are applicable and species t to signify that a slow reaction has been applied. Finally, we add a zero species for all g_i species. Figure 7 shows an example of a k-VG CRN simulating a Course-Rate CRN with $\Lambda = \{a, b\}$ and a fast reaction that deletes a and a slow reaction that deletes b.

▶ **Theorem 23.** *The Void Genesis model is equivalent under polynomial simulation to the Course-Rate model.*

4.5 Step-Cycle CRNs

▶ **Lemma 24.** *Step-Cycle CRNs can simulate any given k-VG CRN under polynomial simulation.*

Construction. Given a k-VG CRN $\mathcal{C}_{\mathcal{KVG}} = ((\Lambda, \Gamma), Z_\emptyset)$, we construct a Step-Cycle CRN $\mathcal{C}_{\mathcal{SC}} = ((\Lambda', \Gamma'), \overrightarrow{S}_0)$. We let $\Lambda' = \Lambda \cup \{G, y, x, w, e_{\lambda_1}, \ldots, e_{\lambda_{|\Lambda_1|}}, z_{\lambda_1}, \ldots, z_{\lambda_{|\Lambda_2|}}\}$ and $\overrightarrow{S}_0 = \vec{y}$. The G species is used to select a reaction, and y is used to check if any non-zero species in Λ have a count of zero. Finally, $e_{\lambda_1}, \cdots, e_{\lambda_{|\Lambda_1|}}$ are used to check the counts of the $\lambda_i \in \Lambda_1$ that are relevant to the simulated reaction, where Λ_1 is the set of non-zero species in Λ. Each reaction $\gamma_i \in \Gamma$ is simulated by Reaction 1 as given in Table 5a. Reactions 2 consumes e species for any reactant whose count does not reach zero. Once the system is terminal, a y is added so as to generate z species corresponding to the remaining reactants using Reaction 3. Another y is added to enable Reaction 4. Once no reaction is applicable in Γ, another y is added to enable Reaction 5, which allows Reaction 6 to execute infinitely. Finally, the system reaches a terminal configuration and prevents the volume from growing infinitely. Figure 8 shows an example of a Step-Cycle CRN simulating a k-VG CRN with $\Lambda = \{a, b\}$ and a reaction that consumes the species a.

Table 5 (a) Reactions for a Step-Cycle CRN to simulate any given k-VG CRN. (b) Reactions for a k-VG CRN to simulate any given Step-Cycle CRN.

$\forall \gamma_i \in \Gamma$:	$1.\vec{G} + \overrightarrow{\mathcal{R}_i} \to \overrightarrow{e}_{\{\mathcal{R}_i\}} + \overrightarrow{\mathcal{P}_i}$
$\forall \lambda_i \in \Lambda_1$:	$2.\vec{\lambda}_i + \vec{e}_{\lambda_i} \to \vec{\lambda}_i$
	$3.\vec{y} + \vec{e}_{\lambda_i} \to \vec{y} + \vec{z}_{\lambda_i}$
	$4.\vec{y} + \vec{y} \to \vec{G}$
	$5.\vec{G} + \vec{y} \leftrightarrow \vec{G} + \vec{w}$
	$6.\vec{w} \leftrightarrow x$

(a) Step-Cycle CRN sim. k-VG CRN.

	$1.\vec{G} + \vec{g}_i \dashrightarrow \vec{r}_1 + \overrightarrow{\mathcal{P}_i}$
$\forall \gamma_i \in \Gamma$, $z_k \in \{\mathcal{R}_i\}$:	$2.\vec{r}_j + \vec{g}_j \to \vec{r}_{j+1} + \vec{g}_j$
	$3.\vec{r}_j + \vec{z}_{g_j} \to \vec{r}_{j+1} + \vec{g}_j$
	$4.\vec{r}_{\lvert\Gamma\rvert+1} \to \vec{G}$
	$5.\vec{G} + \vec{z}_{g_1} + \ldots + \vec{z}_{g_{\lvert\Gamma\rvert}} \to \vec{s}$
	$6.\vec{t} \to \vec{G} + \vec{g}_1 + \ldots + \vec{g}_{\lvert\Gamma\rvert}$
$\forall S_i \in S \setminus \{S_{k-1}\}$:	$7.\vec{s} + \vec{s}_i \dashrightarrow\dashrightarrow \vec{t} + \vec{s}_{i+1} + \overrightarrow{S}_i$
	$8.\vec{s} + \vec{s}_{k-1} \dashrightarrow\dashrightarrow \vec{t} + \vec{s}_0 + \overrightarrow{S}_{k-1}$

(b) k-VG CRN simulating a Step-Cycle CRN.

Table 6 (a) Checking reactants sequentially $\overrightarrow{\mathcal{R}_i} \dashrightarrow \overrightarrow{\mathcal{P}_i}$ (b) Undoing reaction selection if not enough reactants exist for rule i.

	$\vec{G} + \vec{g}_i \to \vec{R}_1^1 + \vec{z}_{g_i}$
$\forall \lambda_j \in \mathcal{R}_i$:	$\vec{R}_i^k + \vec{\lambda}_j \to \vec{R}_i^{k+1} + \vec{\lambda}'_j$
	$\vec{R}_i^{\lvert\mathcal{R}_i\rvert+1} + \overrightarrow{\mathcal{R}_i}' \dashrightarrow\dashrightarrow \vec{r}_1 + \overrightarrow{\mathcal{P}}_i$

(a) Check reactants.

$\forall \lambda_j \in \mathcal{R}_i$:	$\vec{R}_i^k + \vec{z}_{\lambda_j} \to \vec{R}_i^{k^-} + \vec{z}_{\lambda_j}$
	$\vec{R}_i^{k^-} + \vec{z}_{\lambda_j} \to \vec{R}_i^{k-1^-} + \vec{\lambda}_j$
	$\vec{R}_i^{k^-} + \vec{\lambda}'_j \to \vec{R}_i^{k^-} + \vec{\lambda}_j$
	$\vec{R}_1^{1^-} \to \vec{G}$

(b) Undo reactants.

▶ **Lemma 25.** *k-VG CRNs can simulate any given Step-Cycle CRN under polynomial simulation.*

Construction. Given a Step-Cycle CRN $\mathcal{C}_{\mathcal{SC}} = ((\Lambda, \Gamma), (\overrightarrow{S}_0, \overrightarrow{S}_1, \ldots, \overrightarrow{S}_{k-1}))$, we construct a k-VG CRN $\mathcal{C}_{\mathcal{KVG}} = ((\Lambda', \Gamma'), Z_\emptyset)$. We let $\Lambda' = \Lambda \cup \{G, \lambda'_1, \ldots, \lambda'_{|\Lambda|}, z_{\lambda_1}, \ldots, z_{\lambda_{|\Lambda|}}, r_1, \ldots, r_{|\Gamma|+1}, \quad s, s_0, \ldots, s_{k-1}, t\} \cup \{g_i, z_{g_i}, R_i^j, R_i^{j^-}, P_i^l \quad : 1 \leq i \leq |\Gamma|, 1 \leq j \leq |\mathcal{R}_i| + 1, 1 \leq l \leq |\mathcal{P}_i| + 1\}$. The G species is used to select a reaction represented by g_i. The reactants are checked sequentially, converting each reactant into λ'_i. Species r_i reintroduce each consumed g_i into the system. Species s_i represent step i, with species s and t used to transition between steps. Reaction 1 in Table 5b attempts to apply γ_i by checking if enough of each reactant exists, as shown in Table 6. If it is successful, then it simulates the reaction and begins the process of adding $g_1, \ldots, g_{|\Gamma|}$ back into the system (which is carried out in reactions 2, 3, and 4). If a species in γ_i does not exist in the system, then the reactants are reintroduced into the system as shown in Table 6, which also removes g_i from the system, producing z_{g_i}. If there are no executable reactions, then reaction 5 is executed to produce an s species. This s species then reacts with the current "step" of the system, denoted by species s_i. Reaction 7 or 8 then runs, introducing the species corresponding to the step. These reactions also introduce the species t, which then reintroduces G along with each g_i through reaction 6. Figure 9 shows an example of a k-VG CRN simulating a Step-Cycle CRN with $\Lambda = \{a, b\}$ and reaction $a + b \to b$.

▶ **Theorem 26.** *The Void Genesis model is equivalent under polynomial simulation to the Step-Cycle model.*

Extension to deletion-only rules. Given the recent result in [20], these results extend to give the following corollary.

▶ **Corollary 27.** *Even when restricted to at most $(3, 1)$ void rules, the Step-Cycle model is equivalent under polynomial simulation to the Void Genesis model.*

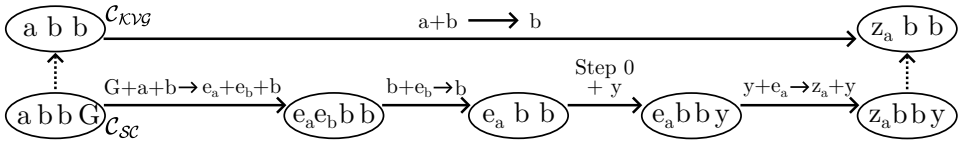


Figure 8 A rule application in the *simulated* k-VG CRN $\mathcal{C}_{\mathcal{KVG}}$ and the equivalent rule application sequence in the *simulating* Step-Cycle CRN $\mathcal{C}_{\mathcal{SC}}$.

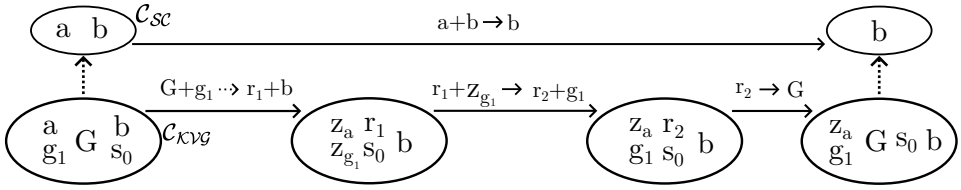


Figure 9 A rule application in the *simulated* Step-Cycle CRN $\mathcal{C}_{\mathcal{SC}}$ and the equivalent rule application sequence in the *simulating* k-VG CRN $\mathcal{C}_{\mathcal{KVG}}$.

Table 7 (a) Reactions for a UI parallel CRN to simulate any given VG CRN. Here, $\overrightarrow{e_{\{\mathcal{R}_i\}}}$ is an e_{λ_j} species created for each of the j different reactants in $\mathcal{R}_i$ (only one e is created if multiple copies of that species are used). (b) Reactions for a kVG CRN to simulate any given UI parallel CRN.

$\forall \gamma_i \in \Gamma:$	$1.\vec{G}+\overrightarrow{\mathcal{R}_i} \to \vec{G_i}+\overrightarrow{e_{\{\mathcal{R}_i\}}}+\overrightarrow{\mathcal{P}_i}$
$\forall \lambda_j \in \Lambda_1:$	$2.\vec{e}_{\lambda_j} \to \vec{r}^1_j + \vec{t}^1_j$
	$3.\vec{r}^1_j + \vec{\lambda}_i \to \vec{r}^2_j$
	$\vec{t}^1_j \to \vec{t}_2$
	$4.\vec{t}_2 + \vec{r}^2_j \to \vec{\lambda}_i + \vec{E}$
	$\vec{t}_2 + \vec{r}^1_j \to \vec{z} + \vec{E}$
	$5.\vec{G_i} + \lvert\{\mathcal{R}_i\}\rvert \cdot \vec{E} \to \vec{G}$

(a) UI parallel CRN simulating a VG CRN.

$\forall \gamma_i \in \Gamma,$ $\forall r_j \in \mathcal{R}_i:$	$\vec{G_i} + \vec{z}_{r_j} \to \vec{N_i} + \vec{X_i} + \vec{z}_{r_j}$ (can not run)
seq.	$\vec{G_i} + \vec{r}_1 \to \vec{R}^2_i$
reacts.	$\vec{R}^j_i + \vec{r}_j \to \vec{R}^{j+1}_i$
	$\vec{R}^j_i + \vec{z}_{r_j} \to \vec{N_i} + \vec{X_i} + \vec{r}_1 + \ldots + \vec{r}_{j-1} + \vec{z}_{r_j}$
$j = \lvert\mathcal{R}_i\rvert$	$\vec{R}^j_i + \vec{r}_j \to \vec{I_i} + \vec{X_i}$ (rule selected)
	$X_1 + \ldots + X_{\lvert\Gamma\rvert} \to \vec{F}$ (rules selected)
$\forall \gamma_i \in \Gamma$	$\vec{I_i} + \vec{F} \to \vec{F_i} + \vec{F} + \overrightarrow{\mathcal{P}}_i$ (output)
	$\vec{N_i} + \vec{F} \to \vec{F_i} + \vec{F}$ (not run)
reset	$\vec{F} + \vec{F_1} + \ldots + \vec{F}_{\lvert\Gamma\rvert} \to \vec{G_i} + \ldots + \vec{G}_{\lvert\Gamma\rvert}$

(b) k-VG simulating UI parallel CRN.

4.6 Unique-Instruction Parallel Model

▶ **Lemma 28.** *UI parallel CRNs can simulate any given VG CRN.*

Construction. Given a Void-Genesis CRN $\mathcal{C}_{\mathcal{VG}} = ((\Lambda, \Gamma), z)$, we construct the UI parallel CRN $\mathcal{C}_{\mathcal{UI}} = (\Lambda', \Gamma')$. We create $\Lambda' = \Lambda \cup \{G, E, t^2\} \cup \{e_{\lambda_j}, t^1_j, r^1_j, r^2_j, G_i : i \in \{1, \ldots, |\Gamma|\}, j \in \{1, \ldots, |\{\mathcal{R}_i\}|\}\}$. The global species G selects a rule $\gamma_i \in \Gamma$ non-deterministically, and produces G_i to disallow Reaction 1 from running again prematurely, a set of species e_λ, and the products of γ_i, as shown in Table 7*a*. The e_λ species then creates a timer species t^i_j, and a checker species r^1_j, which is used to check if a species exists in the system. In Reaction 3, the rules run in parallel, so the timer species goes down, and at the same time, we check if any checker species have incremented. Reaction 4 either reintroduces the consumed species or creates the zero species z, based on the checker species. Both reactions produce the species E, which is later used in Reaction 5 to reintroduce the G species, allowing the system to run another reaction. Figure 10 shows an example of a UI CRN simulating a VG CRN with $\Lambda = \{a, b\}$ and the rule $a + b \to b$. .

▶ **Lemma 29.** *k-VG CRNs can simulate any given UI Parallel CRN.*

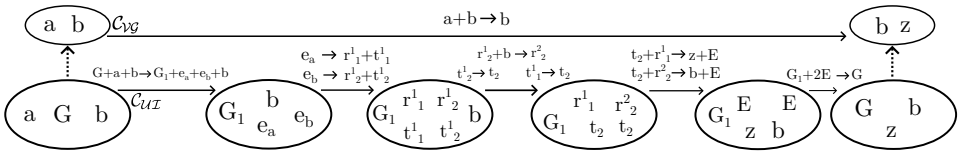


Figure 10 A rule application in the *simulated* void genesis CRN $\mathcal{C}_{\mathcal{VG}}$ and the equivalent rule application sequence in the *simulating* Unique Instruction CRN $\mathcal{C}_{\mathcal{UI}}$.

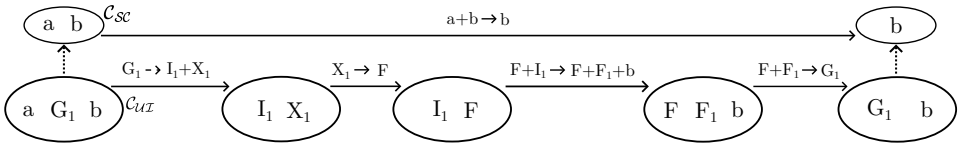


Figure 11 A rule application in the *simulated* Unique Instruction CRN $\mathcal{C}_{\mathcal{UI}}$ and the equivalent rule application sequence in the *simulating* k-void genesis CRN $\mathcal{C}_{\mathcal{KVG}}$.

Construction. Given a UI parallel CRN $\mathcal{C}_{\mathcal{UI}} = (\Lambda, \Gamma)$, we construct a k-Void-Genesis CRN $\mathcal{C}_{\mathcal{KVG}} = ((\Lambda', \Gamma'), Z_\emptyset)$ as described. Let $\Lambda' = \Lambda \cup \{F\} \cup \{G_i, N_i, I_i, F_i, R_i^j : i \in \{1, \ldots, |\Gamma|\}, j \in \{1, \ldots, |\mathcal{R}_i|\}\} \cup \{z_{r_j} : r_j \in \mathcal{R}_i s.t. 1 \leq i \leq |\Gamma|\}$. For each reaction $\gamma_i \in \Gamma$, a G_i species exists which attempts to sequentially consume the reactants of γ_i. If this is successful, it is indicated by the production of an I_i species. If some reactant is missing, it returns any previously consumed reactants and creates an N_i species instead. Both procedures create an X_i species, and when a copy of X_i exists for each reaction, this indicates a maximal set of simulated reactions has been chosen and the F species is created. The N_i or I_i species then turn into a F_i species, as well as produce the products of γ_i if I_i was converted. Once a F_i species exists for each reaction, they combine to reset the rule selection process by creating all the G_i species again. The full reaction set is shown in Table 7*b*, and Figure 11 shows an example of a k-VG CRN simulating a UI CRN with $\Lambda = \{a, b\}$ and reaction $a + b \to b$.

▶ **Theorem 30.** *The Void Genesis model is equivalent under polynomial simulation to the Unique-Instruction parallel model.*

4.7 Register Machine

In this section, we show that Void Genesis CRNs are Turing Universal by constructing a simulation of a standard register machine.

▶ **Theorem 31.** *k-VG CRNs are Turing Universal.*

Proof. Given a Register Machine (RM) with $s_1, \ldots, s_n$ states–each with either a $inc(r_l, s_j)$ or $dec(r_l, s_j, s_k)$ instruction–and $r_1, \ldots, r_m$ registers, we construct the k-VG CRN as follows. Each register and state will be represented by a species, and the instructions will be encoded in the rules. The initial configurations should have one copy of s_1 (assume s_1 is the RM's starting state) and the register species' counts should be equal to the registers they represent.

Table 8 shows what rules should be made for each state of a given RM. Because k-VG CRNs can simulate any given RM, k-VG CRNs are Turing Universal. ◀

Although it can be inferred from this result and the simulation equivalence results that all the models studied here are Turing Universal, we do not give formal proofs of this due to space.

Table 8 Rules for a Void Genesis CRN to simulate a given Register Machine.

Instruction	Relevant Rules	Instruction	Relevant Rules
$s_j : inc(r_i, s_k)$	$\vec{s}_j + \vec{r}_i \to \vec{s}_k + \vec{r}_i + \vec{r}_i$ $\vec{s}_j + \vec{z}_{r_i} \to \vec{s}_k + \vec{r}_i$	$s_j : dec(r_i, s_k, s_l)$	$\vec{s}_j + \vec{r}_i \to \vec{s}_k$ $\vec{s}_j + \vec{z}_{r_i} \to \vec{s}_l + \vec{z}_{r_i}$ $Z_\emptyset(r_i) = z_{r_i}$

5 Conclusion

In this paper, we demonstrate equivalence through polynomial simulation between 5 natural extensions to the CRN model. We centralize these simulations around the Void Genesis CRN model, as this model's ability to detect zero is one of the simplest augmentations to a regular CRN. We then show that Void Genesis CRNs are Turing Universal, implying that Step-Cycle CRNs, Inhibitory CRNs, Parallel CRNs, and Coarse-Rate CRNs are also Turing Universal. While this work is complete in proving equivalence between these models, there are still several interesting open problems to consider (some of which are shown in Figure 1):

- We aim to explore constrained versions of our simulation definition that recover existing notions as special cases. Does restricting the configuration map to be consistent with an underlying species-species mapping immediately results in weak bisimulation? If that underlying map is a total bijective function, does that yield strong bisimulation?
- For iCRNs, a rule is inhibited by the existence of one or more species. Our definition effectively uses a logical `OR` (inhibition is only false when all inhibitor counts are zero). A natural extension is to consider inhibition functions using other logic (e.g., AND – a reaction is inhibited only when all of its inhibitors are present).
- Coarse-Rate CRNs are limited to 2 ranks for reactions. A natural generalization of this model is to allow for k different ranks (k-rate CRNs). What is the relationship between Coarse-Rate CRNs and k-rate CRNs?
- Even when limited to only void reactions (rules where no species are created), step CRNs are able to compute threshold circuits [2, 3]. Does this suggest that Step-Cycle CRNs, even when limited to only void reactions (void Step-Cycle), are still Turing Universal?
- Are there more efficient ways to simulate these augmented CRNs using VG CRNs?
- What is the complexity of reachability in restricted instances of each model, such as [13, 14]? As mentioned, we know the complexity with step-cycles [20], but deletion-only rules have not been explored in detail in the other models.

References

1 Tilak Agerwala. Complete model for representing the coordination of asynchronous processes. Technical report, Johns Hopkins Univ., Baltimore, Md.(USA), 1974.

2 Rachel Anderson, Alberto Avila, Bin Fu, Timothy Gomez, Elise Grizzell, Aiden Massie, Gourab Mukhopadhyay, Adrian Salinas, Robert Schweller, Evan Tomai, and Tim Wylie. Computing threshold circuits with void reactions in step chemical reaction networks. In *10th conference on Machines, Computations and Universality*, MCU'24, 2024.

3 Rachel Anderson, Bin Fu, Aiden Massie, Gourab Mukhopadhyay, Adrian Salinas, Robert Schweller, Evan Tomai, and Tim Wylie. Computing threshold circuits with bimolecular void reactions in step chemical reaction networks. In *International Conference on Unconventional Computation and Natural Computation*, UCNC'24, pages 253–268. Springer, 2024. doi:10.1007/978-3-031-63742-1_18.

4 Marco Antoniotti, Carla Piazza, Alberto Policriti, Marta Simeoni, and Bud Mishra. Taming the complexity of biochemical models through bisimulation and collapsing: theory and practice. *Theoretical Computer Science*, 325(1):45–67, 2004. doi:10.1016/J.TCS.2004.03.064.

5 Rutherford Aris. Prolegomena to the rational analysis of systems of chemical reactions. *Archive for Rational Mechanics and Analysis*, 19(2):81–99, January 1965. doi:10.1007/BF00282276.

6 Bernard Berthomieu and Dmitry A. Zaitsev. Sleptsov nets are turing-complete. *Theoretical Computer Science*, 986:114346, 2024. doi:10.1016/j.tcs.2023.114346.

7 Kim Calabrese and David Doty. Rate-independent continuous inhibitory chemical reaction networks are turing-universal. In *International Conference on Unconventional Computation and Natural Computation*, pages 104–118. Springer, 2024. doi:10.1007/978-3-031-63742-1_8.

8 Matthew Cook, David Soloveichik, Erik Winfree, and Jehoshua Bruck. Programmability of chemical reaction networks. In *Algorithmic bioprocesses*, pages 543–584. Springer, 2009. doi:10.1007/978-3-540-88869-7_27.

9 Wojciech Czerwiński and Łukasz Orlikowski. Reachability in vector addition systems is ackermann-complete. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1229–1240. IEEE, 2022.

10 Qing Dong. A bisimulation approach to verification of molecular implementations of formal chemical reaction networks. *Master's thesis, Stony Brook University*, 2012.

11 Javier Esparza and Mogens Nielsen. Decidability issues for petri nets–a survey. *arXiv preprint arXiv:2411.01592*, 2024. doi:10.48550/arXiv.2411.01592.

12 Jean-Claude Fernandez. An implementation of an efficient algorithm for bisimulation equivalence. *Science of Computer Programming*, 13(2-3):219–236, 1990. doi:10.1016/0167-6423(90)90071-K.

13 Bin Fu, Timothy Gomez, Ryan Knobel, Austin Luchsinger, Aiden Massie, Marco Rodriguez, Adrian Salinas, Robert Schweller, and Tim Wylie. Brief announcement: Reachability in deletion-only chemical reaction networks. In *Proc. of the Symposium on Algorithmic Foundations of Dynamic Networks*, SAND, 2025.

14 Bin Fu, Timothy Gomez, Ryan Knobel, Austin Luchsinger, Aiden Massie, Marco Rodriguez, Adrian Salinas, Robert Schweller, and Tim Wylie. Reachability in deletion-only chemical reaction networks. In *Proc. of the International Conference on DNA Computing and Molecular Programming*, DNA, 2025.

15 Michel Henri Théodore Hack. *Decidability questions for Petri Nets.* PhD thesis, Massachusetts Institute of Technology, 1976.

16 Michel Henri Theódore Hack. Petri net language. Technical report, Massachusetts Institute of Technology, 1976.

17 Robert Johnson, Qing Dong, and Erik Winfree. Verifying chemical reaction network implementations: a bisimulation approach. *Theoretical Computer Science*, 765:3–46, 2019. doi:10.1016/J.TCS.2018.01.002.

18 Richard M Karp and Raymond E Miller. Parallel program schemata: A mathematical model for parallel computation. In *8th Annual Symposium on Switching and Automata Theory (SWAT 1967)*, pages 55–61. IEEE, 1967. doi:10.1109/FOCS.1967.27.

19 Jérôme Leroux. The reachability problem for Petri nets is not primitive recursive. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1241–1252. IEEE, 2022.

20 Austin Luchsinger, Aiden Massie, Robert Schweller, Evan Tomai, and Tim Wylie. Polynomial simulations of crn models with trimolecular void step-cycle crns. In *Proc. of the 22th International Conference on Unconventional Computing and Natural Computing*, UCNC, 2025.

21 Robin Milner. *Communication and concurrency*, volume 84. Prentice hall Englewood Cliffs, 1989.

22 James Lyle Peterson. Petri net theory and the modeling of systems, 1981.

23 Carl Adam Petri. *Communication with automata.* PhD thesis, Technische Universitat Darmstadt, 1966.

24 Phillip Senum and Marc Riedel. Rate-independent constructs for chemical computation. *PloS one*, 6(6):e21414, 2011.

25 Adam Shea, Brian Fett, Marc D Riedel, and Keshab Parhi. Writing and compiling code into biochemistry. In *Biocomputing 2010*, pages 456–464. World Scientific, 2010. URL: `http://psb.stanford.edu/psb-online/proceedings/psb10/shea.pdf`.

26 Seung Woo Shin. *Compiling and verifying DNA-based chemical reaction network implementations*. PhD thesis, California Institute of Technology, 2012.

27 Seung Woo Shin, Chris Thachuk, and Erik Winfree. Verifying chemical reaction network implementations: a pathway decomposition approach. *Theoretical Computer Science*, 765:67–96, 2019. `doi:10.1016/J.TCS.2017.10.011`.

28 Pamela Bridgers Thomas. Petri net: a modeling tool for the coordination of asynchronous processes. Technical report, Tennessee Univ., Knoxville (USA); Oak Ridge Gaseous Diffusion Plant, Tenn.(USA), 1976.

Parameterized Algorithms for the Drone Delivery Problem

Simon Bartlmae
University of Copenhagen, Denmark
Institute of Computer Science, University of Bonn, Germany

Andreas Hene
Institute of Computer Science, University of Bonn, Germany

Joshua Könen
Institute of Computer Science, University of Bonn, Germany

Heiko Röglin
Institute of Computer Science, University of Bonn, Germany

Abstract

Timely delivery and optimal routing remain fundamental challenges in the modern logistics industry. Building on prior work that considers single-package delivery across networks using multiple types of collaborative agents with restricted movement areas (e.g., drones or trucks), we examine the complexity of the problem under structural and operational constraints. Our focus is on minimizing total delivery time by coordinating agents that differ in speed and movement range across a graph. This problem formulation aligns with the recently proposed *Drone Delivery Problem with respect to delivery time* (DDT), introduced by Erlebach et al. [ISAAC 2022].

We first resolve an open question posed by Erlebach et al. [ISAAC 2022] by showing that even when the delivery network is a path graph, DDT admits no polynomial-time approximation within any polynomially encodable factor $a(n)$, unless P=NP. Additionally, we identify the *intersection graph* of the agents, where nodes represent agents and edges indicate an overlap of the movement areas of two agents, as an important structural concept. For path graphs, we show that DDT becomes tractable when parameterized by the treewidth w of the intersection graph, and we present an exact FPT algorithm with running time $f(w) \cdot \text{poly}(n, k)$, for some computable function f. For general graphs, we give an FPT algorithm with running time $f(\Delta, w) \cdot \text{poly}(n, k)$, where Δ is the maximum degree of the intersection graph. In the special case where the intersection graph is a tree, we provide a simple polynomial-time algorithm.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability; Theory of computation → Graph algorithms analysis

Keywords and phrases Complexity, Delivery, FPT algorithms, Graph Theory

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.8

1 Introduction

Drone-based package delivery has the potential to transform the last-mile segment. Recent studies foretell that utilizing unmanned aerial vehicles (i.e., drones) can have a great sustainability impact while meeting increasing demands of customers [13, 17]. Additionally, it is observed that in the U.S. approximately 10–15% of package deliveries are delayed due to increasing loads [7]. According to ElSayed et al. [10], recent surveys indicate that a vast majority of people prefer a same-day delivery, which proves challenging for the logistics industry. Consequently, large companies focus more and more on developing suitable logistics models incorporating drones. Apart from monetary benefits, the utilization of drones can be very helpful in supplying medicine or food in areas struck by war or natural disasters [2].

© Simon Bartlmae, Andreas Hene, Joshua Könen, and Heiko Röglin;
licensed under Creative Commons License CC-BY 4.0

In this work, we study a cooperative delivery setting recently introduced by Erlebach et al. [11] in which drones can hand over packages and work together to achieve the best possible delivery schedule. Each agent operates within a restricted movement area on the graph and is assumed to have unlimited battery capacity. In general, this model is inspired by real-world constraints such as restricted air spaces or environmental circumstances. Restricted movement areas also capture settings with different kinds of agents, not necessarily only drones – different types of agents might be eligible to traverse certain parts of a graph while others might not. Package handover between drones is already implemented in industry; companies such as IBM have developed and patented parcel transfer methods since 2017 [21].

Our focus is on *minimizing total delivery time*, a problem referred to as the *Drone Delivery Problem with respect to time* (DDT) [3, 11]. We study general graphs as well as the special case of path graphs. For path graphs, we first establish a hardness result, followed by the first fixed-parameter tractable algorithms for the path as well as the general settings. For some parameter k, an algorithm is fixed-parameter tractable (FPT) if it runs in time $f(k) \cdot \text{poly}(n)$, where f is a computable function; such algorithms are efficient for small parameters and useful for fine-grained complexity analysis. We analyze the complexity of DDT using the *intersection graph* of the agents, where nodes represent agents and edges indicate overlapping movement areas. The *treewidth* and maximum degree Δ of this graph serve as natural parameters for characterizing instance complexity – especially since DDT is already NP-hard on path graphs. These parameters reflect the extent of agent overlap and potential interference, and are likely to be small in practical settings.

Related Work. For the most part, research regarding collaborative drone delivery focuses on two main objectives: minimizing fuel consumption and minimizing delivery time. In this work, we focus exclusively on the latter. Research on minimizing total consumption differs fundamentally from our analysis, since in our setting there are no battery constraints; therefore, the results do not translate. For work on minimizing fuel consumption, we refer to [5, 11]. Regarding delivery time, Bärtschi et al. [4] and Carvalho et al. [8] show that the problem of delivering a single package can be solved in polynomial time if the agents are free to move without restrictions – even with predetermined starting positions. However, Carvalho et al. [8] also prove that the problem becomes NP-hard if there are at least two packages involved. For the case of restricted movement areas, Erlebach et al. [11] prove that it is NP-hard to approximate DDT within a factor of $O(\min\{n^{1-\varepsilon}, k^{1-\varepsilon}\})$, even if agents have equal speeds and fixed starting positions. Here, n denotes the number of vertices in the graph and k the number of agents. In the problem variant in which the starting positions are not given but can be chosen by the algorithm, they show that no polynomial-time approximation algorithm with any finite approximation ratio exists, unless P=NP. They also show that DDT with fixed starting positions remains NP-hard on path graphs if agents can have arbitrary speeds.

Bartlmae et al. [3] recently refined these results, showing that the problem on a path is still NP-hard if only two different speeds are allowed. This closes the gap regarding speeds as these instances can be solved trivially for only one speed. Additionally, they show that for the special case of unit grid graphs and only two different speeds, the problem remains NP-hard for fixed starting positions and is hard to approximate within a factor of $O(n^{1-\varepsilon})$ for selectable starting positions, as well as agents constrained to movement areas forming rectangles. They also provide a simple n-approximation algorithm for this setting. Complementing these hardness results, Luo et al. [16] demonstrated that on a path with fixed starting positions and exactly two speeds the problem can be solved in polynomial time; the complexity for three or more speeds, however, remains open.

Our Results. In Section 3.1, we provide a hardness result for DDT on path graphs with selectable starting positions, closing open questions posed by Erlebach et al. [11], Bartlmae et al. [3] and Luo et al. [16].

▶ **Theorem 1.** *DDT on a path with selectable starting positions is $a(n)$-APX-hard for any function $a : \mathbb{N} \to \mathbb{R}_+$ that can be computed in polynomial time.*

We then present the first FPT algorithm for this setting in Section 3.2, parameterized by the treewidth of the agents' intersection graph.

▶ **Theorem 2.** *DDT on a path with selectable starting positions can be solved in time $O(2^w \cdot w^2 \cdot k)$, where w denotes the treewidth of the intersection graph.*

In Section 4, we consider the more complex case of general graphs. We design an FPT algorithm whose running time depends on the treewidth and maximum degree of the intersection graph.

▶ **Theorem 3.** *DDT with selectable starting positions can be solved in time $f(\Delta, w) \cdot \text{poly}(n, k)$ on general graphs, where w denotes the treewidth and Δ denotes the maximum degree of the intersection graph and f is a function in w and Δ.*

For the special case where the intersection graph is a tree, we show in Section 4.3 that DDT can be solved in polynomial time. This result is not implied by the previous theorems; it allows for arbitrary maximum degree in the intersection graph.

▶ **Theorem 4.** *DDT on a general graph with selectable starting positions can be solved in time $O(k^2 \cdot n^2)$ if the underlying graph of the intersection graph is a tree.*

2 Preliminaries

In this section, we formally define the Drone Delivery Problem with respect to time (DDT).

Setting. We define an instance of DDT as a tuple $I = (G, (s, t), A)$, where $G = (V, E, \ell)$ with $n = |V|$, is an undirected graph with edge lengths $\ell : E \to \mathbb{R}_{\geq 0}$. The package starting position, as well as its destination, are given by (s, t), where $s \in V$ and $t \in V$. The set A represents all agents, with $k = |A|$. There are two settings regarding agents which are typically considered for DDT:

1. DDT with **selectable positions** (*DDT-SP*): Each agent $a \in A$ is defined as a tuple $a = (v_a, G_a)$, where v_a is the agent's speed and $G_a = (V_a, E_a)$ is a connected subgraph of G representing the agent's movement area. The initial starting position of a can be chosen freely once before delivery is initiated.
2. DDT with **fixed positions** (*DDT-FP*): Each agent $a \in A$ is a tuple $a = (v_a, G_a, p_a)$, with v_a and G_a defined as above, and the additional parameter $p_a \in V_a$ specifies the fixed initial starting position of the agent.

An agent a traverses an edge $\{u, v\} \in E_a$ in time $\frac{\ell(\{u,v\})}{v_a}$, regardless of whether it is carrying the package or not. Whenever two agents meet at the same vertex, they can hand over the package instantaneously. We define the travel time of agent a between two vertices u, v as the length of the shortest path in G_a divided by v_a and denote it by $d_a(u, v)$. If either u or v is not contained in V_a, we define $d_a(u, v) = \infty$. For any pair of vertices and any possible agent the values can be precomputed in polynomial time.

We specify a feasible solution to be a schedule S of consecutive trips by agents delivering the package from s to t. More precisely we want the schedule to consist of two components: A complete edge-by-edge trip of every agent and a complete edge-by-edge trip for the package

specifying the current package carrier at any point in time. For the first part we want for every agent a a set of sorted triples $\mathcal{T}_a$. Every triple (u, v, τ) indicates that agent a moves over edge $\{u, v\}$ starting at time τ. It has to hold that $\{u, v\} \in E_a$ and that for two consecutive triples (u, v, τ) and (u', v', τ') that $u' = v$ and $\tau + \frac{\ell(\{u,v\})}{v_a} \leq \tau'$. Moreover, for DDT-SP we need to provide the starting position p_a for every agent $a \in A$ and for DDT-FP we need to have for the first triple that $u = p_a$. The trip of the package is defined as a sorted set of tuples $\mathcal{T}$ of the form (u, v, a, τ), indicating that agent a carries the package over edge $\{u, v\}$ starting at time τ. A schedule is feasible only if, for every tuple $(u, v, a, \tau) \in \mathcal{T}$, it holds that $(u, v, \tau) \in \mathcal{T}_a$. Again for two consecutive tuples (u, v, a, τ) and (u', v', a', τ') we also require $u' = v$ and $\tau + \frac{\ell(\{u,v\})}{v_a} \leq \tau'$. Additionally, the first tuple must satisfy $u = s$, and the last tuple must satisfy $v = t$. The total delivery time is defined as $c(S) := \tau + \frac{\ell(\{u,v\})}{v_a}$, assuming (u, v, a, τ) is the final tuple in the schedule.

An instance of DDT-SP is illustrated in Figure 1a. Throughout this paper we will focus exclusively on DDT-SP.

Useful Properties. There are several useful properties regarding DDT. Erlebach et al. [11] demonstrated that there exists an optimal schedule in which every involved agent picks up and drops off the package exactly once. The key insight is that instead of using an agent multiple times we let that agent carry the package until its final drop-off without increasing the delivery time.

Another important observation relates to trips made by agents:

▶ **Observation 5.** *In DDT-SP there exists an optimal schedule such that all involved agents move if and only if they carry the package.*

Placing agents initially at the first vertex of their respective trip immediately leads to this observation. Unless stated otherwise, this implicit initial positioning is assumed.

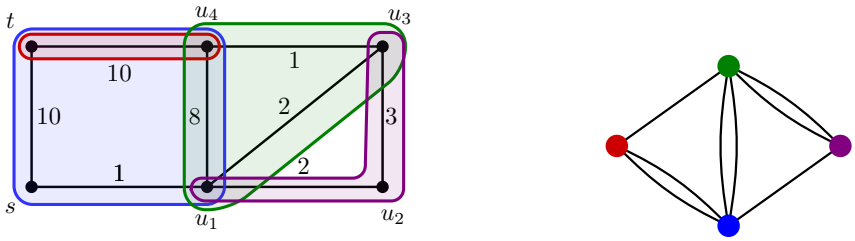
Intersection Graph. A useful structural concept is the *intersection graph* of the agents. We define the intersection (multi)graph G_I of a DDT instance $(G, (s, t), A)$ as follows:

- The vertex set $V(G_I)$ is the set of agents A.
- For any two agents $a, a' \in A$ with corresponding vertex sets V_a and $V_{a'}$ in G, we add $|V_a \cap V_{a'}|$ parallel edges between a and a' in G_I – one for each vertex they share in G. Formally, the edge set $E(G_I)$ is defined as the multiset

$$E(G_I) = \biguplus_{\substack{a,a' \in A \\ a \neq a'}} \{ \underbrace{\{a, a'\}, \ldots, \{a, a'\}}_{|V_a \cap V_{a'}| \text{ times}} \}.$$

Throughout this paper, we treat E as a multiset and omit explicit edge indices where they are not relevant. Intuitively, a feasible schedule corresponds to a path in the intersection graph that starts with an agent intersecting s and ends with an agent intersecting t. If we assume each agent is used at most once, this path will be simple. Figure 1b illustrates the intersection graph for a given instance. We denote by $\Delta(G_I)$ the maximum degree of the intersection graph G_I. When it is clear from the context, we will write Δ instead of $\Delta(G_I)$. For any vertex u, we define B_u as the set of agents that cover u, i.e., $B_u := \{a \in A \mid u \in V_a\}$.

Tree Decomposition. For computing the optimal traversal of the agents, we will use the concept of a *tree decomposition*. A tree decomposition is used as a measure of how similar a given graph is to a tree and was first introduced by Robertson and Seymour [18]. Computing the optimal treewidth is NP-hard [1], but finding the optimal tree decomposition is fixed-parameter tractable if parameterized by treewidth [6].



(a) (b)

Figure 1 Fig. 1a shows a DDT instance with four agents, each restricted to a colored movement area (bounding boxes). The red agent has speed 10, the blue and green agents have speed 1, and the purple agent has speed 2. In the optimal schedule, the blue agent starts at s and delivers the package to u_1 in time 1. The green agent then transports it via u_3 to u_4 in time $2 + 1 = 3$. Finally, the red agent delivers it to the target t in time 1, for a total delivery time of 5. Fig. 1b shows the corresponding intersection graph.

▶ **Definition 6** (Tree Decomposition). *A tree decomposition of a graph $G = (V, E)$ is a pair $\mathcal{T} = (\mathbb{T}, \{X_t\}_{t \in V(\mathbb{T})})$ where $\mathbb{T}$ is a tree with root $r \in V(\mathbb{T})$ and every node $t \in V(\mathbb{T})$ is associated with a set of the vertices $X_t \subseteq V$, called bag. Additionally, the following three conditions must be fulfilled:*

- $\bigcup_{t \in V(\mathbb{T})} X_t = V(G)$, *i.e., every vertex $v \in V(G)$ appears in at least one bag,*
- *for every edge $\{u, v\} \in E(G)$ there must exist a bag $t \in V(\mathbb{T})$ such that the vertices u and v are contained in this bag,*
- *the set of nodes $T_u = \{t \in V(\mathbb{T}) \mid u \in X_t\}$ forms a connected subtree of $\mathbb{T}$ for every choice $u \in V(G)$.*

The *width* of a tree decomposition is the size of its largest bag minus one. The *treewidth* of a graph G is the minimum width over all possible tree decompositions of G, denoted by w. To simplify the algorithm design, we assume the decomposition has a specific structure known as a *nice tree decomposition*: a binary tree with root r and leaves l such that $X_r = \emptyset$ and $X_l = \emptyset$ for all $l \in V(\mathbb{T})$, containing only three types of nodes:

- **Introduce node**: For two nodes $t_{\text{parent}}, t_{\text{child}} \in V(\mathbb{T})$ where t_{parent} has exactly one child t_{child}, t_{parent} is an *introduce node* iff $X_{t_{\text{parent}}} = X_{t_{\text{child}}} \cup \{v\}$ for some $v \in V(G)$.
- **Forget node**: For two nodes $t_{\text{parent}}, t_{\text{child}} \in V(\mathbb{T})$ where t_{parent} has exactly one child t_{child}, t_{parent} is a *forget node* iff $X_{t_{\text{parent}}} \cup \{v\} = X_{t_{\text{child}}}$ for some $v \in V(G)$.
- **Join node**: For three nodes $t_{\text{parent}}, t^1_{\text{child}}, t^2_{\text{child}} \in V(\mathbb{T})$ where t_{parent} has exactly two children t^1_{child} and t^2_{child}, t_{parent} is a *join node* iff $X_{t_{\text{parent}}} = X_{t^1_{\text{child}}} = X_{t^2_{\text{child}}}$.

As in [9] we additionally assume that the edges are also introduced like the vertices in their respective type of node:

- **Introduce edge node**: For every edge $e = \{u, v\} \in E$ there is exactly one introduce edge node $t \in V(\mathbb{T})$ with $u, v \in X_t = X_{t'}$ for its only child bag t' and $X_t = X_{t'}$.

We assume that for every edge $\{u, v\}$ the corresponding introduce edge node appears directly before either u or v is forgotten in an ancestor node.

Using all node types, we define V_t and E_t as the set of vertices and edges that are either contained or introduced in bag t or were introduced in some child of t. Additionally, we define $G_t = (V_t, E_t)$ as the subgraph revealed up to node t. As a consequence for the construction of the introduce edge nodes, for every join node $t \in V(\mathbb{T})$ there are no edges with both endpoints in X_t, i.e. $E_t \cap \{\{u, v\} \mid u, v \in X_t\} = \emptyset$.

Based on a tree decomposition of width w for some graph G, we can always compute a nice tree decomposition (with or without introduce edge nodes) of width w with $O(|V(G)| \cdot w)$ nodes in polynomial time [9].

3 Drone Delivery on Path Graphs

We consider DDT-SP instances where the underlying graph is a path: a sequence of vertices connected in a line, with degree two at each internal vertex and degree one at the two endpoints. For an agent a, we denote its movement area by $[s_a, f_a]$, where s_a and f_a are the leftmost and rightmost vertices that a can access. This setting can also be interpreted as agents moving along a one-dimensional line, with each movement area corresponding to an interval on that line, as illustrated in Figure 2. This allows us to interpret DDT-SP on a path as an instance of the *subinterval covering* problem introduced by Luo et al. [16], where the objective is to select, for each agent's interval, a subinterval such that the union covers $[s, t]$, while minimizing the total cost defined by the length of each selected subinterval divided by the agent's speed.

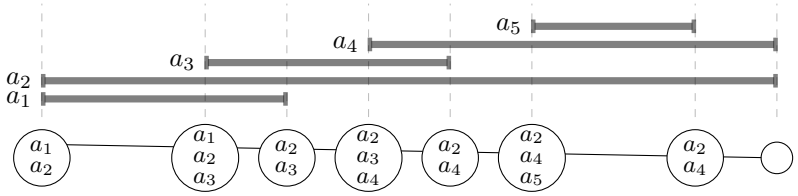


Figure 2 A DDT-SP instance on a line and a corresponding tree decomposition of its intersection graph. At every start or end position we either add or remove the corresponding agent.

For the setting with fixed initial positions, NP-hardness has been proven [3, 11], although strong NP-hardness remains unresolved. In contrast, for the setting with selectable positions – which is the primary focus of this paper – Luo et al. [16] presented a polynomial time algorithm for the case of exactly two speeds; for three or more speeds, however, no results were known prior to our work.

Before presenting parameterized algorithms for DDT on path graphs, we first establish hardness results for this setting, which motivates the relevance and necessity of parameterized approaches.

3.1 Inapproximability

Instead of merely proving NP-hardness, we establish a much stronger result: no polynomial-time algorithm with any polynomially encodable approximation ratio exists, unless P=NP.

▶ **Theorem 1.** *DDT on a path with selectable starting positions is $a(n)$-APX-hard for any function $a : \mathbb{N} \to \mathbb{R}_+$ that can be computed in polynomial time.*

In the following we only present a high-level overview, the complete proof can be found in the full version of the paper. To prove the inapproximability, we reduce from the PartitionInto-k problem.

▶ **Definition 7** (PartitionInto-k)**.** *Given a set of natural numbers $p_1, \ldots, p_n$ and an integer k, the task is to decide whether the index set $\{1, \ldots, n\}$ can be partitioned into k disjoint subsets $I_1 \dot\cup \ldots \dot\cup I_k$ such that $\sum_{i \in I_1} p_i = \cdots = \sum_{i \in I_k} p_i$.*

Note that in this subsection k refers to the number of subsets (and not the agent set). The NP-hardness of PartitionInto-k follows immediately from the hardness of Partition with $k = 2$; however for the proof of Theorem 1 we require the stronger result that PartitionInto-k is also NP-hard in the strong sense, which is included in the full version.

▶ **Theorem 8.** PartitionInto-k *is NP-hard in the strong sense (even with distinct integers).*

This is done by a straightforward reduction from 3-Partition, which was originally shown to be strongly NP-hard by Garey and Johnson [12] and for distinct integers by Hullet et al. [14]. It is particularly important that k is part of the input; for a fixed k, the problem admits an FPTAS, as shown by Sahni [20].

To show that no approximation algorithm $\mathcal{A}$ with a polynomially encodable approximation ratio $a(n)$ can exist (unless P=NP), we demonstrate that such an algorithm could be used to solve PartitionInto-k in polynomial time. We do so by constructing a DDT instance I' from a given PartitionInto-k instance I such that:

I is a "yes"-instance of PartitionInto-k if and only if $c_{\mathcal{A}}(I') \leq d \cdot a(n)$, where $c_{\mathcal{A}}(I')$ is the delivery time of the schedule produced by $\mathcal{A}$ for the instance I' and the value d is determined by the construction.

Let $I = (p_1, \ldots, p_n)$ be an arbitrary PartitionInto-k instance with distinct integers. We assume that the numbers are sorted in decreasing order, i.e., $p_1 > \cdots > p_n$. We denote the total sum of all elements by P.

For each object p_i we create k drones – one for each partition set. We refer to these drones as *element agents* and denote them by $e_{i,j}$, where $e_{i,j}$ is associated with object p_i and partition set $j \in \{1, \ldots, k\}$. The speed of a drone $e_{i,j}$ is denoted by v_j, meaning that all drones associated with the same partition set (i.e., with the same index j) share the same speed. We place several gadgets along the line and aim to ensure that each element agent can only be placed and transport the package at specific designated positions and not at arbitrary locations along the path. To achieve this, we employ two key mechanisms: first, we restrict the movement areas of the agents such that they are clearly unable to assist outside their interval. Second, we choose the ratios between the speeds $v_1, \ldots, v_k$ to be sufficiently large, allowing us to define long intervals that certain agents cannot traverse in their entirety without exceeding the time bound $d \cdot a(n)$.

For each of the k partition sets, we create a *partition gadget*, that enforces that the sum of the weights p_i associated with the element agents $e_{i,j}$ placed at that gadget is at least $\frac{P}{k}$. Placing an element agent $e_{i,j}$ at a partition gadget can thus be interpreted as assigning p_i to the corresponding partition set.

Additionally, for each object p_i we construct a gadget O_i – called an *object gadget* – that ensures each object is assigned to at most one partition set. This is done by requiring that exactly $k - 1$ of the agents $e_{i,1}, \ldots, e_{i,k}$ must be present at the object gadget, resulting in at most one of them being placed at a partition gadget. As a result, we can ensure that each partition gadget ends up with a total weight of exactly $\frac{P}{k}$, if such a partition exists. If not, the construction forces a very large delivery time, causing the bound $d \cdot a(n)$ to be exceeded.

An example layout of the gadgets, ordered as $P_1, \ldots, P_k, O_n, \ldots, O_1$, is shown in Figure 3. Consecutive gadgets are placed sufficiently far apart so that only specially designated fast transition agents can traverse the gaps, ensuring that each element agent can contribute to at most one gadget.

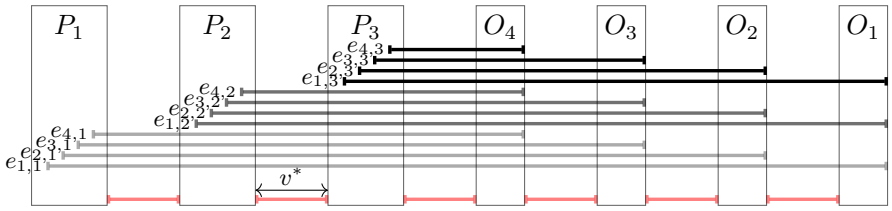


Figure 3 Overview of all gadgets in an example with four objects and three partition sets. The element agents are shown in gray, with darker shades indicating higher speeds. Transition agents (marked in red) with speed $v^* \gg v_1, \ldots, v_k$ are placed between gadgets; if these are bypassed and an element agent is used at multiple gadgets, the time bound $d \cdot a(n)$ is exceeded.

3.2 A Parameterized Algorithm

We now design an FPT algorithm for DDT-SP on path graphs. As a parameter for our approach, we use the maximum overlap – the *thickness* $\sigma := \max_{u \in V(G)} |B_u|$, i.e., the largest number of agents covering any single vertex.

Since each agent corresponds to an interval on the line, the intersection graph of agents is an interval graph. In such graphs, agents covering a common point form a clique, and each maximal clique corresponds to some point on the line [15]. As interval graphs are chordal, their treewidth equals the size of the largest clique minus one [19], which in our case implies $w = \sigma - 1$. Thus, the treewidth of the intersection graph and the thickness parameter are equivalent. This structural property allows for a dynamic programming algorithm over a tree decomposition of the intersection graph. We define the *event points* $e_1 < \cdots < e_{2k}$ as the start and end positions of all agent intervals, i.e., the multiset $\{s_a, f_a \mid a \in A\}$. We sort these events from left to right along the path $(e_1 < \cdots < e_{2k})$, resolving ties by placing start points before end points; the order among start points or among end points is chosen arbitrarily. In an optimal schedule, handovers occur only at these event points – specifically at $s_{a'}$ or f_a for a handover from a to a' – since otherwise extending the faster agent's segment would yield a strictly better or equally good schedule.

We interpret each event e_i as a node in a tree decomposition that forms a path, see Fig. 2. We can consider the set of agents covering e_i, Z_{e_i} as the bag at position i. If e_i is the start vertex of some agent a, we say that a is *introduced* to the bag, and if e_i is the end point of some agent a, it is *forgotten*. We define a dynamic program over the path decomposition induced by this event sequence. At each e_i, we maintain:

- the *bag* Z_{e_i} of agents covering e_i,
- a subset $S \subseteq Z_{e_i}$ of agents already used,
- a current carrier $a \in Z_{e_i}$ responsible for transporting the package from e_i to e_{i+1}.

Since at most σ agents cover any point, we have $|Z_{e_i}| \leq \sigma$, and the number of subsets $S \subseteq Z_{e_i}$ (representing already-used agents) is bounded by 2^σ. The dynamic programming table stores the minimum delivery time to reach e_i under the assumption that agent a will deliver it to e_{i+1} and the current set of already used agents is S, denoted by $D[e_i, S, a]$. We assume no agent starts at t or ends at s, as those movement intervals can be cut off without affecting the solution. Furthermore, at most one handover occurs per event point, and it happens only if $e_i = s_{a'}$ or $e_i = f_a$, as argued before. While we only store delivery times in the DP table, the full delivery sequence can be reconstructed via backtracking.

In the following we always assume for some DP entry $D[e_i, S, a]$ that $a \in S$, otherwise the entry is defined as ∞. Assume we have already computed all possible entries $D[e_{i-1}, S \subseteq Z_{e_{i-1}}, a \in Z_{e_{i-1}}]$ correctly, and now wish to compute $D[e_i, S \subseteq Z_{e_i}, a \in Z_{e_i}]$.

The formal correctness of the following lemmas appears in the full version of the paper.

Introduce Event. For an introduce event e_i, it is the start point of an agent a', so a' is now newly contained in Z_{e_i}. We differentiate between a' being included into S, which corresponds to the package being handed over to a' and the case where a' is omitted (for now). Note that when a' enters S it will also be the dedicated package carrier.

▶ **Lemma 9.** *Let e_i be an introduce event that introduces some agent a'. Assume that for all valid combinations the solutions $D[e_{i-1}, S' \subseteq Z_{e_{i-1}}, a \in S']$ have been computed correctly, then we correctly compute $D[e_i, S \subseteq Z_{e_i}, a \in S]$ in time $O(\sigma)$.*

Forget Event. For a forget event e_i, an end point for some agent a' was reached, thus a' is removed from Z_{e_i}. We consider all possibilities for how a' could have been involved in an optimal delivery: Either a' delivered the package to e_i and now hands it over to a different agent a, or a' was involved in the optimal delivery but did not carry the package to e_i, or a' never held the package at any point. Out of all of these options we again take the solution variant with the smallest cost.

▶ **Lemma 10.** *Let e_i be a forget event that forgets some agent a'. Assume that for all valid combinations the solutions $D[e_{i-1}, S' \subseteq Z_{e_{i-1}}, a \in S']$ have been computed correctly, then we correctly compute $D[e_i, S \subseteq Z_{e_i}, a \in S]$ in time $O(1)$.*

▶ **Theorem 2.** *DDT on a path with selectable starting positions can be solved in time $O(2^w \cdot w^2 \cdot k)$, where w denotes the treewidth of the intersection graph.*

Proof. The correctness follows from the correctness of the different types of events and that the optimal solution must be stored in some entry $D[e_j, S \subseteq Z_{e_j}, a]$ for some final end point event $e_j = e_{2k}$. Since for every type of event we have runtime at most $O(\sigma)$ and the set of instances at an event point e_i is upper bounded by $2^\sigma \cdot \sigma$ and $2k$ event points exist, we get a final runtime of $O(2^\sigma \cdot \sigma^2 \cdot k) = O(2^w \cdot w^2 \cdot k)$. ◀

4 Fixed-Parameter Tractability on General Graphs

Before designing an FPT algorithm on general graphs, we argue that, w.l.o.g., the intersection graph can be assumed to be simple – i.e., agents intersect at most once. Any instance can be transformed accordingly, increasing degree and treewidth by only a constant.

4.1 Unique Intersections between Agents

Given any DDT instance $(G, (s, t), A)$, we now create an instance $(G', (s', t'), A')$ that has equivalent schedules, but its intersection graph G'_I is simple and therefore any pair of agents overlap in at most one vertex. Each agent $a \in A$ originally moves on a (possibly overlapping) subgraph G_a of G. To guarantee that no pair of agents share multiple vertices, we create a *disjoint copy* of every G_a and relabel each vertex u as (u, a). The k original agents are then confined to their respective copies G'_a. Whenever two subgraphs G_a and G_b share a vertex u in G, the transformed graph G' now contains two distinct copies, i.e., (u, a) and (u, b). We connect these copies of the same vertex by edges (of length 0) and introduce a dedicated *auxiliary agent* h_u with speed 1 and movement area restricted to the newly added edges of length 0 between the copies. Thus, the package can be transferred instantly between any two copies of the same original vertex, while original agents share no common vertex. The

resulting instance preserves all feasible schedules with the same delivery time as the original one but it also satisfies the new constraint that any pair of agents shares at most one vertex, as any pair of original agents a', b' does not share a common vertex in G', neither does any pair of helping agents $h_u, h_{u'}$, and agents a' and h_u may only intersect in (u, a). In the full version of the paper, we formally define such a transformation and prove the following lemma.

▶ **Lemma 11.** *For any DDT instance $(G, (s,t), A)$, there exists an equivalent instance $(G', (s', t'), A')$, where $w(G'_I) \leq w(G_I) + 1$, $\Delta(G'_I) \leq \Delta(G_I) + 1$ and G'_I is simple.*

4.2 Tree Decomposition Algorithm for Unique Intersections

Let $(G, (s,t), A)$ be a drone delivery instance in which each pair of agents $a, a' \in A$ intersects in at most one vertex. For an agent $u \in A$, we define $N_u := \{v \in V(G_I) \mid \{u, v\} \in E(G_I)\}$. Let $D : A \times A \to V(G) \cup \{-\}$ be a function that returns for two agents a_i, a_j their unique intersection point $V_i \cap V_j$ if it exists and $-$ otherwise. Let G_I be the intersection graph, and assume we have a nice tree decomposition $\mathcal{T} = (\mathbb{T}, \{X_t\}_{t \in V(\mathbb{T})})$ of G_I of width w and maximum degree $\Delta = \max_{u \in V(G_I)} |N_u|$.

We now design a dynamic program over $\mathcal{T}$ that is parameterized by w and Δ and computes the optimal agent tour. The algorithm is inspired by standard treewidth-based DP techniques for TSP[1]. For each node $t \in V(\mathbb{T})$, we process the subgraph $G_t = (V_t, E_t)$.

A delivery tour can be modeled as an ordered sequence $(a_1, a_2, \ldots, a_b)$ of agents. Each contiguous subsequence $A_{i,j} = (a_i, \ldots, a_j)$ induces a valid subpath in G_t, if the corresponding edges $\{a_l, a_{l+1}\} \in E(G_t)$ for all $l \in [i, j-1]$, and $a_i, a_j \in X_t$. Intermediate agents a_l with $i < l < j$ incur cost $d_{a_l}(D(a_{l-1}, a_l), D(a_l, a_{l+1}))$.

However, to determine the cost of agents a_i, a_j we require knowledge about the corresponding unknown preceding and succeeding agents to determine entry and exit points. Thus, the DP state additionally stores for the boundary agents in X_t their assumed predecessor and successor. This allows consistent cost computation when merging subpaths at join nodes.

We slightly modify the input instance to streamline boundary cases: we add two artificial agents a_s and a_t, each covering only vertex s and t respectively, and having zero speed. The unique intersection property extends naturally to these agents. We now additionally require any valid delivery tour to start with a_s and end with a_t. Both agents are required to be always contained in any bag, increasing the treewidth by exactly 2. Note that the actual delivery can still start at s, since a_s can immediately hand over the package at no cost.

To simplify boundary handling, we also introduce auxiliary agents a'_s and a'_t, each identical to a_s and a_t, but always assumed to lie outside the tree decomposition (i.e., they are never contained in any bag). A delivery tour will now implicitly require the tour to start with a'_s and end with a'_t. Together, this guarantees that for each boundary agent, which always includes a_s and a_t, its predecessor and successor are explicitly known, enabling us to compute exact cost for partial solutions.

Each DP state $D[t, B_t^0, B_t^1, B_t^2, M, \hat{A}, \hat{Z}]$ represents the optimal forest of paths in subgraph G_t that connects agents according to the following:

- $B_t^i \subseteq X_t$ for $i \in \{0, 1, 2\}$ contains all agents with degree exactly i
- $M \subseteq B_t^1 \times B_t^1$ encodes directed pairwise path endpoints as a matching
- $\hat{A}$ and $\hat{Z}$ assign known predecessor/successor agents to agents in B_t^1:
  - $\hat{A}$ maps to a neighbor agent s.t. the corresponding edge is not contained in E_t
  - $\hat{Z}$ maps to the neighbor s.t. the corresponding edge is contained in E_t

[1] See e.g. `http://www.cs.bme.hu/~dmarx/papers/marx-warsaw-fpt2`, page 17.

Each path P_j in a feasible forest F must satisfy the following conditions:

- The endpoints of P_j lie in B_t^1 and every agent $a \in B_t^i$ has degree i in F
- For every pair $(a, a') \in M$, there must be a path P_j in F with both of these as endpoints

The cost of a forest F is then evaluated using the previously mentioned reasoning of computing the cost of every individual agent measured by its two neighboring agents in the corresponding sequence. For convenience we will sometimes represent functions as set of tuples, i.e. $\hat{A} = \{(u, \hat{A}(u)) \mid u \in V(G_t)\}$.

For the root node r the DP entry $D[r, \emptyset, \{a_s, a_t\}, \emptyset, \{(a_s, a_t)\}, \{(a_s, a'_s), (a_t, a'_t)\}, \hat{Z}]$ captures the optimal path from a_s to a_t, if $\hat{Z}$ correctly encodes the adjacent agents to a_s and a_t in an optimal sequence. By iterating over all valid candidates for the neighbors $u \in N_{a_s} \setminus \{a'_s\}$ and $v \in N_{a_t} \setminus \{a'_t\}$ we determine the optimal solution.

To compute the delivery cost of a sequence of agents, we use the predecessor/successor assignment $\hat{A}$. For an agent a of degree 1 we write $\hat{A}(a)$ to denote the predecessor (if a is the start of a path) or successor (if a is the end), and its other adjacent agent by $\hat{Z}(a)$ and the partner of a in the matching M by $M(a)$. We will refer to $\hat{A}(a)$ as the predecessor/successor partner (or *psp* for short) and to $\hat{Z}(a)$ as the next/previous interior partner (or *npip*).

For a feasible forest $S = \{S_1, \ldots, S_b\} \subseteq E_t$, each component $S_i = (a_1, \ldots, a_\ell)$ corresponds to a path of agents starting and ending at agents in B_t^1. Using $\hat{A}$, we extend each such path to the sequence $\hat{S}_i = (\hat{a}_0, a_1, \ldots, a_\ell, \hat{a}_{\ell+1})$, where $\hat{a}_0 = \hat{A}(a_1)$ and $\hat{a}_{\ell+1} = \hat{A}(a_\ell)$. These agents define the handover points at the start and end of the path that involves agents a_1 and a_ℓ, which is necessary to compute the full cost contribution of each agent.

The cost of the extended sequence $\hat{S}_i$ is given by $c(\hat{S}_i) = \sum_{j=1}^{l_i} d_{\hat{a}_j}(D(\hat{a}_{j-1}, \hat{a}_j), D(\hat{a}_j, \hat{a}_{j+1}))$. Setting the predecessor of a_s as a'_s and the successor of a_t as a'_t ensures sequences starting at a_s or ending at a_t are correctly computed as well. In this case we will incur an additional cost of 0, since these agents must immediately handover the package at their respective vertex.

The total cost of the solution S is the sum over all component sequences, i.e., $c(S) := \sum_{i=1}^{b} c(\hat{S}_i)$. In the following we write $\hat{A}(S_i)$ for the extended sequence $\hat{S}_i$. Finally, for any solution $S \subseteq E(G_I)$ and node $t \in V(\mathbb{T})$ we define $S[t] := S \cap E_t$ and say that S *induces* the instance $(t, B_t^0, B_t^1, B_t^2, M, \hat{A}, \hat{Z})$ if

- each $a \in B_t^i$ has degree i in $S[t]$,
- each pair $(u, v) \in M$ is connected by a path in $S[t]$ with $\hat{A}(u), \hat{A}(v)$ as their psp agents, and
- for each npip agent a on a path, the adjacent agent corresponds to $\hat{Z}(a)$.

For an example instance and corresponding agent degrees, see Figure 4. In the following arguments we will for simplicity assume that any DP entry contains the actual forest. If each DP entry only contains its cost one can, by simple recursive pointer-logic, compute the optimal solution in the end recursively. Therefore, we will argue correctness of each entry by considering the actual forest, but derive the computational runtime as if only the cost is saved in every DP entry.

The following lemma shows that indeed it is enough to prove correctness for every subinstance in the DP to derive the optimal solution. The proof is provided in the full version of the paper.

▶ **Lemma 12.** *For every node $t \in V(\mathbb{T})$ and optimal tour $\mathrm{OPT} \subseteq E(G_I)$ that starts with agent a_s and ends with a_t, the partial solution $\mathrm{OPT}[t]$ is optimal for the induced instance.*

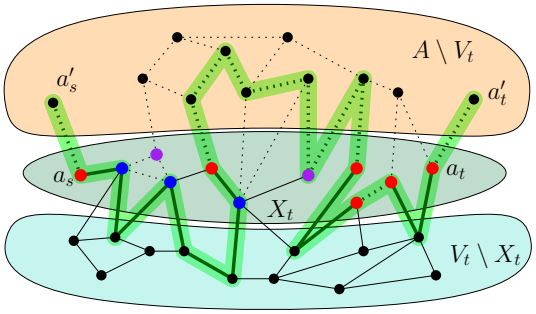


Figure 4 Example instance G_I with corresponding sets of intersections between agents. Agents a_s, a_t are always contained in X_t and a'_s, a'_t are always contained in $A \setminus V_t$. Possible solution ordering of agents shown in green. Red agents have degree 1 in V_t, blue agents have degree 2 and violet agents degree 0. Dashed lines correspond to edges not introduced so far.

We now describe how to handle the different node types in the dynamic program. We will assume that the instance to solve corresponds to a valid specification, otherwise we assign it cost ∞. The proof of correctness for the different node types can be found in the full paper.

Leaf node: At a leaf node t, only a_s, a_t are present and no edges exist. The only valid forest is the empty one with cost 0, so we set $D[t, B_t^0, B_t^1, B_t^2, M, \hat{A}, \hat{Z}] = \{\emptyset\}$.

Introduce node: Let t be an introduce node with child t' and $X_t = X_{t'} \cup \{v\}$. Since v has no incident edges at this point, it must have degree 0 in any valid solution, i.e., $v \in B_t^0$. Therefore v does not influence any solution derived for t' and we set $D[t, B_t^0, B_t^1, B_t^2, M, \hat{A}, \hat{Z}] = D[t', B_t^0 \setminus \{v\}, B_t^1, B_t^2, M, \hat{A}, \hat{Z}]$.

▶ **Lemma 13.** *Let node t introduce an agent v, and let t' be its child. Assume that for all valid combinations the sets $D[t', B_t^{0\prime}, B_t^{1\prime}, B_t^{2\prime}, M', \hat{A}', \hat{Z}']$ have been computed, then we correctly compute $D[t, B_t^0, B_t^1, B_t^2, M, \hat{A}, \hat{Z}]$ in time $O(1)$.*

Forget node: Let t be a forget node with child t' and $X_t = X_{t'} \setminus \{v\}$. Since $v \notin X_t$ and $v \neq a'_s, a'_t$, it must have degree either 0 or 2 in any feasible solution, depending on whether v was used in the respective optimal solution. Therefore, we check both corresponding entries at the child node t' and choose the one with minimum cost.

▶ **Lemma 14.** *Let node $t \in V(\mathbb{T})$ remove some vertex v, and let t' be its child. Assume that for all valid combinations, the sets $D[t', B_t^{0\prime}, B_t^{1\prime}, B_t^{2\prime}, M', \hat{A}', \hat{Z}']$ have been computed, then we correctly compute $D[t, B_t^0, B_t^1, B_t^2, M, \hat{A}, \hat{Z}]$ in time $O(1)$.*

Introduce edge node: Let t be an introduce-edge node with child t' and let $e = \{u, v\}$ be the newly introduced edge. If e is not used in the optimal solution, we reuse the solution from t'. Otherwise, we consider the instance where the degrees of u and v are decreased by 1 and consider all valid choices for their next potential npip agent via $\hat{Z}$. Additionally, we update M accordingly to reflect all possible new pairings that are possible for solutions containing e.

▶ **Lemma 15.** *Let node $t \in V(\mathbb{T})$ introduce an edge $e = \{u, v\}$, and let t' be its child. Assume that for all valid combinations, the sets $D[t', B_t^{0\prime}, B_t^{1\prime}, B_t^{2\prime}, M', \hat{A}', \hat{Z}']$ have been computed, then we correctly compute $D[t, B_t^0, B_t^1, B_t^2, M, \hat{A}, \hat{Z}]$ in time $O((w+1) \cdot \Delta^2)$.*

Join node: Let t be a join node with children t_1 and t_2. To compute an optimal solution, we combine compatible solutions from both children while ensuring degrees, matching, psp agents and npip agents correctly align, and take the solution with minimal cost.

▶ **Lemma 16.** *If node $t \in V(\mathbb{T})$ is a join node with two children $t_1, t_2 \in V(\mathbb{T})$ and we correctly computed all solutions for all child instances for nodes t_1 and t_2, then we correctly compute $D[t, B_t^0, B_t^1, B_t^2, M, \hat{A}, \hat{Z}]$ in time $O((w+2)^{O(\Delta+w)})$.*

Using these lemmata, we establish the final theorem, with the complete proof included in the full version of this work.

▶ **Theorem 3.** *DDT with selectable starting positions can be solved in time $f(\Delta, w) \cdot \text{poly}(n, k)$ on general graphs, where w denotes the treewidth and Δ denotes the maximum degree of the intersection graph and f is a function in w and Δ.*

4.3 A Polynomial-Time Algorithm for Intersection Trees

We now briefly describe an exact algorithm for computing optimal schedules when the intersection graph of the underlying graph forms a tree. While this may seem similar to the setting previously considered, it does not follow directly, as the treewidth may be constant, but the vertex degrees can be unbounded.

We first show how to compute an optimal solution for any instance, given a fixed order of agents $(a_1, \ldots, a_b)$ in the optimal schedule.

For this, we build a layered directed graph G' whose *layers* represent successive handovers:

- Layer 0: single vertex v_s representing the source s.
- Layer i $(1 \leq i \leq b)$: one vertex for every intersection point of the movement areas of a_i and a_{i+1} (for $i = b$ the region of a_{b+1} is the sink region containing t).
- Layer $b+1$: single vertex v_t for the destination t.

Edges connect every vertex in layer $i-1$ to every vertex in layer i; the weight equals the travel time of agent a_i between these two points. Thus each s–t path in G' selects exactly one handover point per layer and has length equal to the total delivery time for that sequence of drones. Conversely, any schedule that uses these agents in this particular order defines such a path. Hence, the shortest path in G' yields the optimal schedule for the given order, which can easily be computed.

▶ **Lemma 17.** *Given any ordered sequence of agents $(a_1, \ldots, a_b)$, we can compute the optimal route to deliver the package from s to t using the agents in this order in $O(b^2 \cdot n^2)$.*

Proof. We construct a directed weighted delivery graph $G' = (V', E')$, $w : E' \to \mathbb{R}_{\geq 0}$. The vertex set V' consists of $b+1$ layers. In layer 0 we have a vertex v_{a_1}, in layer $b+1$ we have a vertex v_t and in layer $i \in [b]$ we have all intersection points between agent a_i and a_{i+1}, i.e., $V_{a_i} \cap V_{a_{i+1}}$. In the following, we refer to these vertices as $V_i = \{v_1^i, \ldots, v_{m_i}^i\}$. We connect v_{a_1} with every vertex v_j^1 for $j \in [m_1]$ and set its weight as $d_{a_1}(s, v_j^1)$. This connection represents the case that agent a_1 transports the package from s to agent a_2 via some intersection point v_j^1. Similarly, we connect a vertex v_j^i , $j \in [m_i]$ in layer i with a vertex $v_{j'}^{i+1}$, $j' \in [m_{i+1}]$, if there exists a path in G_{a_i} that connects v_j^i with $v_{j'}^{i+1}$. For each such edge we again set its weight to the time it takes to get from the first intersection point to the next using agent a_i, i.e., we set the weight of the edge as $d_{a_i}(v_j^i, v_{j'}^{i+1})$. At last, we set the connections from the second last layer b to v_t in the same manner, i.e., we set its weight to $d_{a_b}(v_j^b, t)$, where $j \in [m_b]$.

We want to argue, that the shortest path in this graph has to correspond to an optimal delivery tour of the agents in the specified order. Let $P_{\text{OPT}} \subseteq E$ be the optimal tour and $(p_1^{\text{OPT}} = s, p_2^{\text{OPT}}, \ldots, p_b^{\text{OPT}}, p_{b+1}^{\text{OPT}} = t)$ be the set of vertices at which an exchange of the package happens, including s and t. We can see that for each section $p_i^{\text{OPT}}, p_{i+1}^{\text{OPT}}$, the delivery

happens using agent a_i to agent a_{i+1}. Since the delivery points are specified by $p_i^{\text{OPT}}, p_{i+1}^{\text{OPT}}$, we can upper bound the optimal travel time of a_i for this section by $d_{a_i}(p_i^{\text{OPT}}, p_{i+1}^{\text{OPT}})$. At last, for the final section $p_b^{\text{OPT}}, p_{b+1}^{\text{OPT}} = t$, the delivery time of agent a_b is upper bounded by $d_{a_b}(p_b^{\text{OPT}}, p_{b+1}^{\text{OPT}} = t)$.

We can see that the optimal path $(p_1^{\text{OPT}} = s, p_2^{\text{OPT}}, \ldots, p_b^{\text{OPT}}, p_{b+1}^{\text{OPT}} = t)$ is always a feasible shortest $s - t$ path solution in graph G', and since its cost is at most the cost of the optimal solution we know that the cost of the shortest path tour will be at most the shortest delivery time in the original instance when using the agents in the specified order. On the other hand, every feasible path in our graph corresponds to a unique tour using the same set of agents in the same order and exchanging the package at vertices $p_1 = s, p_2, \ldots, p_b, p_{b+1} = t$. If one such tour would have delivery time strictly less than OPT, this would contradict its optimality assumption, since the vertices on the shortest path also define a feasible drone delivery routing that delivers the package from s to t using the same set of agents in the given order. Therefore it follows, that the shortest path gives us the optimal delivery tour.

Our graph G' consists of at most $(n-1) \cdot b + 2$ vertices and at most $2 \cdot n + (b-2) \cdot n^2$ edges. Using Dijkstra's algorithm with lists, which has runtime $O(n^2)$, we get a final runtime of $O(b^2 \cdot n^2)$. ◀

One could use this property to calculate the optimal schedule for general graphs in time $O(k \cdot k! \cdot k^2 \cdot n^2)$. It also allows us to solve DDT-SP efficiently, if the underlying simple graph of the intersection graph is a tree, in time $O(k^2 \cdot n^2)$. We conclude by proving Theorem 4.

▶ **Theorem 4.** *DDT on a general graph with selectable starting positions can be solved in time $O(k^2 \cdot n^2)$ if the underlying graph of the intersection graph is a tree.*

Proof. Any delivery schedule corresponds to a path in the intersection graph. If this graph is a tree, there always exists a unique simple path between any two agents in G_I. Given the first and last agents used in an optimal schedule, the order of all participating drones can be determined by this path, as each agent appears at most once. Using this order and Lemma 17, we can efficiently compute the delivery schedule. To identify the first and last agents, we consider all valid combinations of agents that can pick up the package at s and drop it off at t. Since each vertex in the graph belongs to at most two agents – otherwise, a clique of size at least three would be present in G_I, contradicting the tree property – there are at most four such combinations. Simply enumerating all four candidate pairs and evaluating each leads to an overall runtime of $O(k^2 \cdot n^2)$. ◀

5 Concluding Remarks and Future Work

We first proved that the Drone Delivery Problem with selectable starting positions (DDT-SP) is $a(n)$–APX-hard even on path graphs, thereby resolving an open question posed by Erlebach et al. and Bartlmae et al. [3,11]. On the algorithmic side, we showed that DDT-SP on a path is fixed-parameter tractable when parameterized solely by the treewidth w of the agents' intersection graph. Employing more sophisticated techniques and parameterizing by both w and the maximum degree Δ of the intersection graph, we further obtained an FPT algorithm for general graphs. When the intersection graph is a tree, a layered-graph formulation yields a polynomial-time exact algorithm.

We believe that the introduction of the intersection graph and its properties is therefore of great value when analyzing the complexity of DDT.

The question of whether DDT on a path is strongly NP-hard remains open for both DDT-FP and DDT-SP. For DDT-SP on a path the complexity for any fixed number of speeds greater than two also remains unresolved. Future work could tighten our parameter

dependencies, or establish stronger lower bounds such as W-hardness. Another promising direction is to analyze DDT under smoothed and semi-random input models, where small random perturbations of worst-case instances provide a more realistic measure of expected computational complexity and can guide the development of practical algorithms.

References

1 Stefan Arnborg, Derek G Corneil, and Andrzej Proskurowski. Complexity of Finding Embeddings in a k-Tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987. doi:10.1137/0608024.

2 Dane Bamburry. Drones: Designed for Product Delivery. *Design Management Review*, 26(1):40–48, 2015. doi:10.1111/drev.10313.

3 Simon Bartlmae, Andreas Hene, and Kelin Luo. On the Hardness of the Drone Delivery Problem. In Irene Finocchi and Loukas Georgiadis, editors, *Algorithms and Complexity - 14th International Conference, CIAC 2025, Rome, Italy, June 10-12, 2025, Proceedings, Part II*, volume 15680 of *Lecture Notes in Computer Science*, pages 200–215. Springer, 2025. doi:10.1007/978-3-031-92935-9_13.

4 Andreas Bärtschi, Daniel Graf, and Matús Mihalák. Collective Fast Delivery by Energy-Efficient Agents. In Igor Potapov, Paul G. Spirakis, and James Worrell, editors, *43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018*, volume 117 of *LIPIcs*, pages 56:1–56:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.MFCS.2018.56.

5 Lotte Blank, Kien C. Huynh, Kelin Luo, and Anurag Murty Naredla. Algorithms for the Collaborative Delivery Problem with Monitored Constraints. In Shin-ichi Nakano and Mingyu Xiao, editors, *WALCOM: Algorithms and Computation - 19th International Conference and Workshops on Algorithms and Computation, WALCOM 2025, Chengdu, China, February 28 - March 2, 2025, Proceedings*, volume 15411 of *Lecture Notes in Computer Science*, pages 62–78. Springer, 2025. doi:10.1007/978-981-96-2845-2_5.

6 Hans L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. In S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal, editors, *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 226–234. ACM, 1993. doi:10.1145/167088.167161.

7 Heleen Buldeo Rai, Sara Verlinde, and Cathy Macharis. Unlocking the failed delivery problem? Opportunities and challenges for smart locks from a consumer perspective. *Research in Transportation Economics*, 87:100753, 2021. E-groceries, digitalization and sustainability. doi:10.1016/j.retrec.2019.100753.

8 Iago A. Carvalho, Thomas Erlebach, and Kleitos Papadopoulos. On the fast delivery problem with one or two packages. *Journal of Computer and System Sciences*, 115:246–263, 2021. doi:10.1016/J.JCSS.2020.09.002.

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 Mo ElSayed, Ahmed Foda, and Moataz Mohamed. The impact of civil airspace policies on the viability of adopting autonomous unmanned aerial vehicles in last-mile applications. *Transport Policy*, 145:37–54, 2024. doi:10.1016/j.tranpol.2023.10.002.

11 Thomas Erlebach, Kelin Luo, and Frits C. R. Spieksma. Package Delivery Using Drones with Restricted Movement Areas. In Sang Won Bae and Heejin Park, editors, *33rd International Symposium on Algorithms and Computation, ISAAC 2022, December 19-21, 2022, Seoul, Korea*, volume 248 of *LIPIcs*, pages 49:1–49:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ISAAC.2022.49.

12 M. R. Garey and David S. Johnson. Complexity Results for Multiprocessor Scheduling under Resource Constraints. *SIAM J. Comput.*, 4(4):397–411, 1975. doi:10.1137/0204035.

13 Anne Goodchild and Jordan Toy. Delivery by drone: An evaluation of unmanned aerial vehicle technology in reducing CO2 emissions in the delivery service industry. *Transportation Research Part D: Transport and Environment*, 61:58–67, 2018. Innovative Approaches to Improve the Environmental Performance of Supply Chains and Freight Transportation Systems. doi:10.1016/j.trd.2017.02.017.

14 Heather Hulett, Todd G. Will, and Gerhard J. Woeginger. Multigraph realizations of degree sequences: Maximization is easy, minimization is hard. *Oper. Res. Lett.*, 36(5):594–596, 2008. doi:10.1016/J.ORL.2008.05.004.

15 Cornelis Lekkeikerker and Johan Boland. Representation of a finite graph by a set of intervals on the real line. *Fundamenta Mathematicae*, 51(1):45–64, 1962.

16 Kelin Luo, Chenran Yang, Zonghan Yang, and Yuhao Zhang. The subinterval cover problem. In Vincent Chau, Christoph Dürr, Minming Li, and Pinyan Lu, editors, *Frontiers of Algorithmics - 19th International Joint Conference, IJTCS-FAW 2025, Paris, France, June 30 - July 2, 2025, Proceedings*, volume 15828 of *Lecture Notes in Computer Science*, pages 152–165. Springer, 2025. doi:10.1007/978-981-96-8312-3_12.

17 Almodather Mohamed and Moataz Mohamed. Unmanned Aerial Vehicles in Last-Mile Parcel Delivery: A State-of-the-Art Review. *Drones*, 9(6), 2025. doi:10.3390/drones9060413.

18 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.

19 Donald J. Rose, Robert Endre Tarjan, and George S. Lueker. Algorithmic Aspects of Vertex Elimination on Graphs. *SIAM J. Comput.*, 5(2):266–283, 1976. doi:10.1137/0205021.

20 Sartaj Sahni. Algorithms for Scheduling Independent Tasks. *J. ACM*, 23(1):116–127, 1976. doi:10.1145/321921.321934.

21 Frank Schroth. IBM granted patent for package transfer between drones. https://dronelife.com/2017/04/30/ibm-granted-patent-package-transfer-drones/, 2017. [Accessed 23-06-2025].

Small Space Encoding and Recognition of k-Palindromic Prefixes

Gabriel Bathie
DIENS, École normale supérieure de Paris, PSL Research University, France
LaBRI, Université de Bordeaux, France

Jonas Ellert
DIENS, École normale supérieure de Paris, PSL Research University, France

Tatiana Starikovskaya
DIENS, École normale supérieure de Paris, PSL Research University, France

Abstract

Palindromes are non-empty strings that read the same forward and backward. We study the problem of recognizing so-called *k-palindromic* strings, which can be represented as the concatenation of exactly k palindromes. [Rubinchik and Shur, MFCS 2020] showed that the problem is solvable in linear space and time. We present a read-only algorithm that recognizes all k-palindromic prefixes of a string T of length n in $\mathcal{O}(n \cdot 6^{k^2} \cdot \log^k n)$ time and $\mathcal{O}(6^{k^2} \cdot \log^k n)$ space. As a corollary, we also obtain a read-only algorithm for computing the palindromic length of T, i.e., the smallest k such that T is k-palindromic, in $\mathcal{O}(n \cdot 6^{k^2} \cdot \log^{\lceil k/2 \rceil} n)$ time and $\mathcal{O}(6^{k^2} \cdot \log^{\lceil k/2 \rceil} n)$ space.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases palindromic length, read-only algorithms, palindromes

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.9

Related Version *Full Version*: `https://doi.org/10.48550/arXiv.2410.03309` [5]

Funding *Gabriel Bathie, Jonas Ellert, and Tatiana Starikovskaya:* Partially funded by grant ANR-20-CE48-0001 from the French National Research Agency.

1 Introduction

A *palindrome* is a non-empty string that equals its reversed copy, i.e., a string that reads the same both forward and backward. Throughout, we denote the language of palindromes by PAL. Natural variants of PAL include the language of even-length palindromes PAL_{ev} and the language of palindromes of length greater than one $\mathsf{PAL}_{>1}$. Recognising $\mathsf{PAL}^*_{\text{ev}}$ (often referred to as "palstar"), $\mathsf{PAL}^*_{>1}$, and $\mathsf{PAL}^k = \{P_1P_2 \cdots P_k : P_i \in \mathsf{PAL}, 1 \leq i \leq k\}$ is a classical problem of formal language theory, introduced by Knuth, Morris, and Pratt [27].[1] In this problem, one is given an input string and must decide whether it is in the language.

Languages $\mathsf{PAL}^*_{\text{ev}}$, $\mathsf{PAL}^*_{>1}$, and PAL^k are context-free, and Valiant's parser from 1975 recognizes them in $\mathcal{O}(n^\omega)$ time, where n is the length of the input and ω is the matrix multiplication exponent. Only in 2018, Abboud, Backurs, and Vassilevska Williams showed that Valiant's parser is optimal if the current clique algorithms are optimal [1], meaning that for general context-free languages, there is little hope of achieving a faster recognition algorithm. The origins of the study of derivatives of the PAL languages are in fact in line with the result of [1]: At one time, it was popularly believed that $\mathsf{PAL}^*_{\text{ev}}$ cannot be recognised in linear time, and it was considered as a candidate for a "hard" context-free language (see

[1] * is a Kleene star.

© Gabriel Bathie, Jonas Ellert, and Tatiana Starikovskaya;
licensed under Creative Commons License CC-BY 4.0

[27, Section 6]). However, [27] refuted this hypothesis by showing an $\mathcal{O}(n)$-time recognition algorithm for $\mathsf{PAL}^*_{\text{ev}}$. Manacher [32] found another way to recognize $\mathsf{PAL}^*_{\text{ev}}$ in linear time, and Galil [16] derived a real-time recognition algorithm (see also Slisenko [35]). Later, Galil and Seiferas [17] showed a linear-time recognition algorithm for $\mathsf{PAL}^*_{>1}$.

Recognition of PAL^k appeared to be a much more intricate problem. Galil and Seiferas [17] succeeded to design linear-time recognition algorithms for the cases $k = 1, 2, 3, 4$, but the general question remained open for almost 40 years. Only in 2015, Kosolobov, Rubinchik, and Shur [29] showed an $\mathcal{O}(nk)$-time recognition algorithm for PAL^k for all $k \in \mathbb{N}^+$, which was finally improved to optimal $\mathcal{O}(n)$ time by Rubinchik and Shur in 2020 [34]. A related question is that of computing the *palindromic length* of a string T, which is defined to be the smallest integer k such that $T \in \mathsf{PAL}^k$. The first $\mathcal{O}(n \log n)$-time algorithms for computing the palindromic length were presented in [12, 25, 33]. Finally, Borozdin et al. [10] showed an optimal $\mathcal{O}(n)$-time algorithm for this problem.

Our contributions. In this work, we turn our attention to the *space complexity* of recognising PAL^k and computing the palindromic length. We start by presenting a characterization of prefixes of a given string that belong to PAL^k. For $k = 1$, we refer to these prefixes as *prefix-palindromes*, and otherwise as *k-palindromic prefixes*. A crucial component of the linear time algorithm by Borozdin et al. [10] is the following property: the prefix-palindromes of a length-n string can be expressed as $\mathcal{O}(\log n)$ sets of form $\{XQ^a : a \in \{1, \dots, u\}\}$, where u is an integer. In order to encode k-palindromic prefixes, we introduce a notion of *affine prefix sets* of order k. Intuitively, such a set consists of prefixes of the form $XQ_1^{a_1}Q_2^{a_2}\cdots Q_k^{a_k}$ with $\forall i \in [1, k] : a_i \in \{1, \dots, u_i\}$, where u_i are integers. That is, rather than a single repeating substring Q, we allow multiple different substrings Q_i of different lengths. An affine prefix set of order k can then be encoded in $\mathcal{O}(k)$ space. By carefully analyzing the rich structure of periodic substrings induced by k-palindromic prefixes, we show that they can be expressed by a small number of affine prefix sets.

▶ **Theorem 1.1.** *Let $0 < \epsilon < 1$ be constant, $T[1..n]$ a string, and $k \in \mathbb{N}^+$. The set of prefixes of T that belong to PAL^k is the union of $\mathcal{O}(6^{k^2/(2-\epsilon)} \cdot \log^k n)$ affine prefix sets of order $\leq k$.*

The remainder of the paper focuses on the main ideas behind Theorem 1.1. In the full version [5], apart from explaining all details, we show a lower bound for the size of the representation (Theorem 1.2), and provide read-only algorithms for computing affine prefix sets and palindromic length (Theorems 1.3 and 1.4). The lower bound shows that our representation is within a $\log n$ factor of being asymptotically optimal for constant k. It is obtained by constructing a large family of strings uniquely identifiable by their palindromic prefixes.

▶ **Theorem 1.2.** *Let $T[1..n]$ be a string and let $k \in \mathbb{N}^+$. Encoding the lengths of the prefixes of T that belong to PAL^i, for each $i \in [1, k]$, requires $\Omega(k^{-k} \cdot (\log_3 n)^k)$ bits of space.*

The read-only algorithms directly implement the techniques used to show Theorem 1.1.

▶ **Theorem 1.3.** *Let $0 < \epsilon < 1$ be constant. Given a string $T[1..n]$ and $k \in \mathbb{N}^+$, there is a read-only algorithm that returns a compressed representation of all prefixes of T that belong to PAL^i, for each $i \in [1, k]$, in $\mathcal{O}(n \cdot 6^{k^2/(2-\epsilon)} \cdot \log^k n)$ time and $\mathcal{O}(6^{k^2/(2-\epsilon)} \cdot \log^k n)$ space.*

▶ **Theorem 1.4.** *Given a string $T[1..n]$, there is a read-only algorithm that computes the palindromic length k of T in $\mathcal{O}(n \cdot 6^{k^2} \cdot \log^{\lceil k/2 \rceil} n)$ time and $\mathcal{O}(6^{k^2} \cdot \log^{\lceil k/2 \rceil} n)$ space.*

In particular, for $k = \mathcal{O}(\log \log n)$, the algorithm uses $n \log^{\mathcal{O}(k)} n$ time and $\log^{\mathcal{O}(k)} n$ space, and for $k = o(\sqrt{\log n})$, it uses $n^{1+o(1)}$ time and sublinear $n^{o(1)}$ space. In the regime of small palindromic length, this is an improvement over all previously-known algorithms [10, 34], which require $\Omega(n)$ space. It remains an intriguing open problem to achieve *both* linear time and sublinear space. On the other hand, Theorem 1.2 does not imply a lower bound for the algorithms of Theorems 1.3 and 1.4 because they have access to the input. This said, proving an $\Omega(\log^{f(k)} n)$ space lower bound for such algorithms appears out of reach with current techniques. The only lower bound method for read-only string processing the authors are aware of relies on deterministic branching programs [9], and shows that any sublinear-space algorithm for longest common substring requires slightly superlinear time [28].

Related work. The problem of computing palindromes in small space has received significant attention: the longest palindromic substring [8, 24], and all approximate prefix-palindromes [3, 6] have been studied. More broadly, small-space recognition of formal languages has been explored for regular [19, 20, 21, 22, 11], Dyck [26, 30, 31], visibly pushdown [2, 14, 18, 7], context-free [23], and $\mathsf{DLIN}/\mathsf{LL}(k)$ languages [4].

2 Preliminaries

Series, strings, and substrings. For $i, j \in \mathbb{Z}$, we write $[i, j] = [i, j+1) = (i-1, j] = (i-1, j+1)$ to denote $\{h \in \mathbb{Z} \mid i \le h \le j\}$. A series $a_1, b_1, c_1, a_2, b_2, c_2, \ldots, a_t, b_t, c_t$ is denoted by $(a_i, b_i, c_i)_{i=1}^{t}$. The empty series is denoted by ε. We use the dot-product to denote the concatenation of two series, e.g., for $t \ge 3$ one can represent $(a_i, b_i, c_i)_{i=1}^{t} = (a_i, b_i, c_i)_{i=1}^{t-3} \cdot (a_i, b_i, c_i)_{i=t-2}^{t}$. We may omit the subscript and superscript for series of length one, e.g., $(a_1, b_1, c_1) = (a_i, b_i, c_i)_{i=1}^{1}$.

A *string* T of length $|T| = n$ is a sequence of n *symbols* from a set Σ, which we call the *alphabet*. The input string is also called *the text*. We denote the set of all length-n strings by Σ^n, and we set $\Sigma^{\le n} = \bigcup_{m=0}^{n} \Sigma^m$ as well as $\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n$. The empty string is denoted by ε. For $i, j \in [1, n]$, the i-th symbol in T is denoted by $T[i]$. The *substring* $T[i..j] = T[i..j+1) = T(i-1..j] = T(i-1..j+1)$ is the empty string ε if $j < i$, and the string $T[i]T[i+1]\cdots T[j]$ otherwise. We may call a substring $T[i..j]$ a *fragment* of T to emphasize that we mean the specific occurrence of $T[i..j]$ that starts at a position i. For example, in the string $T = \texttt{abcabc}$, the substrings $T[1..3]$ and $T[4..6]$ are identical, but $T[1..3]$ and $T[4..6]$ are distinct fragments. A string S is a *prefix* of T if there is $i \in [1, n]$ such that $S = T[1..i]$, in which case we may simply write $T[..i]$. Similarly, S is a *suffix* of T if there is $i \in [1, n]$ such that $S = T[i..n]$, in which case we may simply write $T[i..]$. We extend this notion to the empty suffix $T[n+1..n] = T[n+1..]$ and the empty prefix $T[1..0] = T[..0]$. A substring (hence also a suffix or prefix) of T is *proper* if it is shorter than T. When introducing a string S, we may simply say that $S[1..m]$ is a string rather than saying that S is a string of length m. The concatenation of two strings $S[1..m]$ and $T[1..n]$ is the string $S[1]S[2]\cdots S[m]T[1]T[2]\cdots T[n]$, denoted by either $S \cdot T$ or simply ST. For non-negative integer a, we write T^a to denote the length-(an) string obtained by concatenating a copies of T. We extend this idea to non-negative rational exponents $\alpha \in \mathbb{Q}$, for which we write T^α to denote $T^{\lfloor \alpha \rfloor} \cdot T[1..(\alpha n \bmod n)]$. We only use this notation if $\alpha n \in \mathbb{N}$.

Palindromes and periodicities. For a string $T[1..n]$, we write $\mathsf{rev}(T)$ to denote its reverse, i.e., $\mathsf{rev}(T) = T[n]T[n-1]\cdots T[1]$. We then say that T is a palindrome if and only if T is non-empty and $T = \mathsf{rev}(T)$. The set of all palindromes is denoted by PAL. For a positive

integer k, the set PAL^k contains all the strings that can be written as the concatenation of exactly k palindromes. We refer to such strings as *k-palindromic*. If $k = 1$, and a string is a one-palindromic prefix of another string, we also refer to it as *prefix-palindrome*.

We define the *forward cyclic rotation* $\mathsf{rot}(T) = T[2..n]T[1]$. More generally, a cyclic rotation $\mathsf{rot}^s(T)$ with *shift* $s \in \mathbb{Z}$ is obtained by iterating rot (if s is positive) or the inverse operation rot^{-1} (if s is negative) exactly $|s|$ times. A non-empty string $T[1..n]$ 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. If a string X can be represented as Y^a for some primitive string Y and integer a, then Y is called *the primitive root* of X.

A string $T[1..n]$ has *period* $p \in \mathbb{N}^+$ if $\forall i \in [1, n-p] : T[i] = T[i+p]$, or equivalently if $T[1..n-p] = T(p..n]$. The string $T[1..n-p] = T(p..n]$ is a *border* of T. If T has period $p \leq n/2$, then we say that T is p-periodic. If T has period $p \leq n$, then it can be written as $T = P^{\lfloor n/p \rfloor} P[1..n \bmod p]$, where $P = T[1..p]$. We may alternatively use a rational exponent and write $T = P^{n/p}$. Below, we provide some simple auxiliary lemmas regarding periodic strings and palindromes (with proofs in the full version).

▶ **Fact 2.1** (Periodicity Lemma [13]). *If p and q are distinct periods of a string of length at least $p + q - \gcd(p, q)$, then $\gcd(p, q)$ is a period of the string.*

▶ **Corollary 2.2** (Folklore). *For a primitive string Q, the minimal period of Q^2 is $|Q|$.*

▶ **Lemma 2.3.** *Assume that a palindrome P has a q-periodic prefix of length $m \geq 3q/2$. If $|P| \leq 2m - q$, then P is q-periodic.*

Model of computation. We assume the word-RAM model of computation [15], using words of size $\Theta(\log n)$ when processing an input string of length n. The presented algorithms are deterministic and read-only, i.e., they cannot write to the memory occupied by the input. Space complexities are stated in number of words, ignoring the space occupied by the input.

3 Combinatorial Properties of Affine Prefix Sets

In this section, we study the combinatorial structure of k-palindromic prefixes of T. We start with the definition of *affine sets*, which we will use as a scaffolding for our analysis.

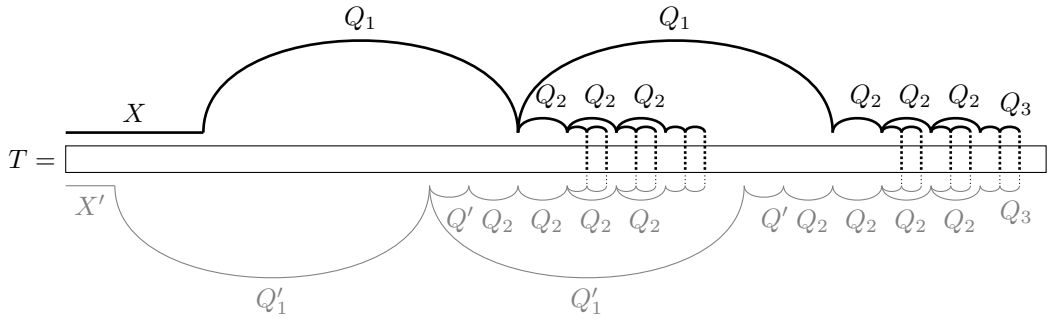
▶ **Definition 3.1** (Affine sets). *A set of strings $\mathcal{A}$ is* affine *if there exist $t \in \mathbb{N}_0$, a string X, primitive strings $Q_1, \ldots, Q_t$, and positive integers $\ell_1, \ldots, \ell_t$ and $u_1, \ldots, u_t$ such that*

$$\forall i \in [1, t] : \ell_i \leq u_i \qquad \text{and} \qquad \mathcal{A} = \{XQ_1^{a_1} \cdots Q_t^{a_t} \mid \forall r \in [1, t] : a_r \in [\ell_r, u_r]\}.$$

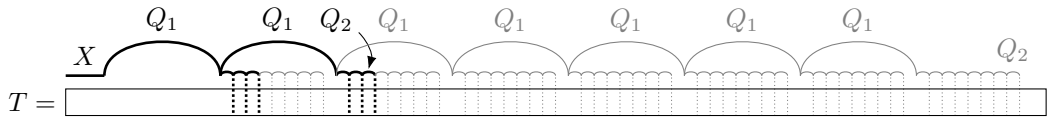
The tuple $\langle X, (Q_i, \ell_i, u_i)_{i=1}^t \rangle$ is a representation *of $\mathcal{A}$, and t is the* order *of the representation. The order of $\mathcal{A}$ is the minimal order achieved by any of its representations. We call $\{Q_i\}$* the components *of a representation, and ℓ_i (resp., u_i) the exponent lower (resp., upper) bounds.*

A representation *generates* (the strings of) the corresponding affine string set. If $\langle X, (Q_i, \ell_i, u_i)_{i=1}^t \rangle$ generates $\mathcal{A}$ and $\langle X', (Q'_i, \ell'_i, u'_i)_{i=1}^{t'} \rangle$ generates $\mathcal{B}$, then their concatenation is defined as $\langle X, (Q_i, \ell_i, u_i)_{i=1}^t \cdot (Y, a, a) \cdot (Q'_i, \ell'_i, u'_i)_{i=1}^{t'} \rangle$, where Y is a primitive string and a is a positive integer such that $Y^a = X'$ (i.e., Y is the primitive root of X'). The concatenation generates $\mathcal{A} \cdot \mathcal{B} = \{A \cdot B : A \in \mathcal{A} \wedge B \in \mathcal{B}\}$. (If $X' = \varepsilon$, then the concatenation is $\langle X, (Q_i, \ell_i, u_i)_{i=1}^t \cdot (Q'_i, \ell'_i, u'_i)_{i=1}^{t'} \rangle$.)

In what follows, we consider *affine prefix sets*, i.e., affine sets that contain only prefixes of the given input string T. We will show that a small number of affine prefix sets suffices to represent the k-palindromic prefixes of T. The case where $k = 1$, i.e., the structure of



(a) An affine prefix set $\mathcal{A}$ of a string T with representation $\langle X, (Q_1, ,) \cdot (Q_2, ,) \cdot (Q_3, ,)\rangle$ (drawn above T). This representation is irreducible. The set $\mathcal{A}$ contains all the prefixes of T that end at positions drawn in dotted lines. In this example, the set $\mathcal{A}$ has the alternative representation $\langle X', (Q'_1, 1, 2) \cdot (Q', 1, 1) \cdot (Q_2, 2, 4) \cdot (Q_3, 1, 2)\rangle$. This representation is reducible because Q' has the same exponent upper and lower bound, and because Q_2 has an exponent lower bound larger than 1.



(b) An affine prefix set $\mathcal{A}$ of a string T with representation $\langle X, (Q_1, 1, 2) \cdot (Q_2, 1, 3)\rangle$ (drawn in black). If this representation is strongly affine, then its expansion $\langle X, (Q_1, 1, 7) \cdot (Q_2, 1, 8)\rangle$ is also a representation of an affine prefix set of T (drawn in gray).

Figure 1 Affine prefix sets.

prefix-palindromes, is well-understood: there are $\mathcal{O}(\log n)$ groups of such palindromes, where each group can be expressed as an arithmetic progression and a corresponding periodic prefix of T (see [10, Lemma 5]). Below, we restate this result in the framework of affine prefix sets.

▶ **Lemma 3.2.** *The prefix-palindromes of a string $T[1..n]$ can be partitioned into $\mathcal{O}(\log n)$ affine sets of order at most 1. Each set of order 1 has representation $\langle U(VU)^\ell, (VU, 1, u)\rangle$ for some $U \in \mathsf{PAL} \cup \{\varepsilon\}$, $V \in \mathsf{PAL}$ and integers $\ell \geq 1$ and $u > 1$.*

3.1 Reducing affine prefix sets

A single affine set may have multiple equivalent representations. For example, the affine set $S = \{\mathtt{caba}, \mathtt{cababa}, \mathtt{cabababa}\}$ is represented by $\langle \mathtt{c}, (\mathtt{ab}, 1, 3)$ and $(\mathtt{a}, 1, 1)\rangle$, $\langle \mathtt{ca}, (\mathtt{ba}, 1, 3)\rangle$. Arguably, the latter representation is preferable, as it has a lower order and can thus be encoded more efficiently. Hence we propose a way of potentially decreasing the order of a representation by *reducing* it.

▶ **Definition 3.3** (Irreducible representation). *A representation $\langle X, (Q_i, \ell_i, u_i)_{i=1}^t\rangle$ of an affine string set is* irreducible *if and only if $\forall r \in [1, t] : 1 = \ell_r < u_r$ and $\forall r \in [1, t) : |Q_r| > |Q_{r+1}|$.*

From now on, we say that Q_r with $r \in [1, t]$ is *fixed* if $\ell_r = u_r$, and *flexible* otherwise. If there is some $r \in [1, t)$ such that $|Q_r| \leq |Q_{r+1}|$, then we say that there is an *inversion* between Q_r and Q_{r+1}. Thus, a representation is irreducible if and only if all components are flexible and have unit lower bounds, and there are no inversions. As per this definition, $\langle \mathtt{ca}, (\mathtt{ba}, 1, 3)\rangle$ is the only irreducible representation of S from the previous example. (See also Figure 1a.)

Properties of flexible components. Now we show how to make an arbitrary representation irreducible, possibly decreasing (but never increasing) its order. The reduction exploits the structure of periodic substrings induced by flexible components.

▶ **Lemma 3.4.** *Let $\langle X, (Q_i, \ell_i, u_i)_{i=1}^t \rangle$ be a representation of an affine prefix set, and consider any $r \in [1, t)$ such that Q_r is flexible. Then $|Q_r|$ is a period of every string $Q_r^{a_r} Q_{r+1}^{a_{r+1}} \cdots Q_t^{a_t}$ that satisfies $a_r \in \mathbb{N}_0$ and $\forall j \in (r, t] : a_j \in [\ell_j, u_j]$.*

Proof. Let $P = Q_1^{\ell_1} Q_2^{\ell_2} \cdots Q_r^{\ell_r}$ and $S = Q_{r+1}^{a_{r+1}} Q_{r+2}^{a_{r+2}} \cdots Q_t^{a_t}$. By the definition of an affine prefix set, XPS is a prefix of the underlying string T. Since Q_r is flexible, it holds $\ell_r < u_r$, and thus XPQ_rS is also a prefix of T. If both XPS and XPQ_rS are prefixes of T, then S is a prefix of Q_rS. Hence Q_rS and S have periods $|Q_r|$. Consequently, $Q_r^{a_r} S$, for all $a_r \in \mathbb{N}_0$, also has period $|Q_r|$. ◀

If two adjacent components Q_r and Q_{r+1} are flexible, then the lemma allows us to obtain the following lower bound on the length of Q_r.

▶ **Lemma 3.5.** *Let $\langle X, (Q_i, \ell_i, u_i)_{i=1}^t \rangle$ be a representation of an affine prefix set, and let $r \in [1, t)$. If both Q_r and Q_{r+1} are flexible, then either $Q_r = Q_{r+1}$ or*

$$|Q_r| > |Q_{r+1}^{u_{r+1}-1}| + \left(\sum_{j=r+2}^{t} |Q_j^{u_j}| \right) + \gcd(|Q_r|, |Q_{r+1}|).$$

Proof. For flexible Q_r and Q_{r+1}, let $q_r = |Q_r|$, $q_{r+1} = |Q_{r+1}|$, and $p = \gcd(q_r, q_{r+1})$. Let $Q'_{r+1} = Q_{r+1}^{u_{r+1}} Q_{r+2}^{u_{r+2}} \cdots Q_t^{u_t}$. By Lemma 3.4, both q_r and q_{r+1} are periods of Q'_{r+1}, and q_r is a period of $Q_r Q'_{r+1}$. Since q_r is a period of $Q_r Q'_{r+1}$, it is also a period of $Q_r Q_{r+1}$. Hence $Q_r = Q_{r+1}$ if and only if $q_r = q_{r+1}$. For the sake of contradiction, assume that the lemma does not hold, i.e., $q_r \neq q_{r+1}$ and $q_r \leq |Q'_{r+1}| - q_{r+1} + p$. We make two observations.

First, Q'_{r+1} is of length $|Q'_{r+1}| \geq q_r + q_{r+1} - p$, and it has distinct periods q_r and q_{r+1}. The periodicity lemma (Fact 2.1) implies that p is a period of Q'_{r+1}, and thus also a period of its prefix Q_{r+1}. If $p < q_{r+1}$, then $Q_{r+1} = Q_{r+1}[1..p]^{q_{r+1}/p}$, which contradicts the primitivity of Q_{r+1}. Second, Q'_{r+1} is of length $|Q'_{r+1}| \geq q_r + q_{r+1} - p \geq q_r$. Since q_r is a period of $Q_r Q'_{r+1}$, we know that Q_r is a prefix of Q'_{r+1}. Hence p is also a period of Q_r. If $p < q_r$, then $Q_r = Q_r[1..p]^{q_r/p}$, which contradicts the primitivity of Q_r.

We have shown that $\gcd(q_r, q_{r+1}) \geq \max(q_r, q_{r+1})$. This is only possible if $\gcd(q_r, q_{r+1}) = q_r = q_{r+1}$, which contradicts the assumption that $q_r \neq q_{r+1}$. Therefore, the lemma holds. ◀

▶ **Lemma 3.6.** *Let $\langle X, (Q_i, \ell_i, u_i)_{i=1}^t \rangle$ be an irreducible representation of an affine prefix set $\mathcal{A}$ of a string of length n. Then it holds $|\mathcal{A}| = \prod_{i=1}^t u_i$ and $t \leq \log_2 n$.*

Proof. Let $E = \{(a_i)_{i=1}^t \mid \forall i \in [1, t] : a_i \in [1, u_i]\}$ of cardinality $|E| = \prod_{i=1}^t u_i$ be the set of exponent configurations admitted by the representation (where $[1, u_i] = [\ell_i, u_i]$ because the representation is irreducible). Then $\mathcal{A} = \{XQ_1^{a_1} Q_2^{a_2} \cdots Q_t^{a_t} \mid (a_i)_{i=1}^t \in E\}$. In order to show $|\mathcal{A}| = |E|$, it suffices to show that no two elements in E generate the same string.

For the sake of contradiction, assume that there are distinct sequences $(a_i)_{i=1}^t, (b_i)_{i=1}^t \in E$ that generate the same string $S = XQ_1^{a_1} Q_2^{a_2} \cdots Q_t^{a_t} = XQ_1^{b_1} Q_2^{b_2} \cdots Q_t^{b_t}$. Let $r \in [1, t]$ be the minimal index such that $a_r \neq b_r$, and assume w.l.o.g. that $a_r > b_r$. Then S has the prefix $XQ_1^{a_1} \cdots Q_{r-1}^{a_{r-1}} Q_r^{b_r} = XQ_1^{b_1} \cdots Q_{r-1}^{b_{r-1}} Q_r^{b_r}$, and we can factorize the corresponding suffix in two different ways as $Q_r^{a_r - b_r} Q_{r+1}^{a_{r+1}} \cdots Q_t^{a_t} = Q_{r+1}^{b_{r+1}} \cdots Q_t^{b_t}$. However, the two factorizations have different lengths $|Q_r^{a_r - b_r} Q_{r+1}^{a_{r+1}} \cdots Q_t^{a_t}| > |Q_r Q_{r+1}| > |Q_{r+1}^{u_{r+1}} \cdots Q_t^{u_t}| \geq |Q_{r+1}^{b_{r+1}} \cdots Q_t^{b_t}|$, where the second inequality is due to Lemma 3.5. Because of this contradiction, there cannot be distinct sequences $(a_i)_{i=1}^t, (b_i)_{i=1}^t \in E$ that define the same string.

Finally, it holds $\forall i \in [1,t] : u_i \geq 2$ for any irreducible representation, and thus $|\mathcal{A}| = \prod_{i=1}^{t} u_i \geq 2^t$. Since trivially $|\mathcal{A}| \leq n$, it follows $2^t \leq n$ or equivalently $t \leq \log_2 n$. ◀

Transforming representations. Now we use the properties of flexible components to transform an arbitrary representation into an irreducible one. We use the following operations.

▶ **Lemma 3.7.** *Let $\rho = \langle X, (Q_i, \ell_i, u_i)_{i=1}^{t} \rangle$ be a representation of an affine prefix set $\mathcal{A}$.*

1. *If there is $r \in [1,t)$ such that Q_r is flexible and Q_{r+1} is fixed, then let $y = |Q_{r+1}^{\ell_{r+1}}| \bmod |Q_r|$. $\mathcal{A}$ has representation*

$$\mathsf{switch}_r(\rho) = \langle X, (Q_i, \ell_i, u_i)_{i=1}^{r-1} \cdot (Q_{r+1}, \ell_{r+1}, u_{r+1}) \cdot (\mathsf{rot}^y(Q_r), \ell_r, u_r) \cdot (Q_i, \ell_i, u_i)_{i=r+2}^{t} \rangle.$$

2. *If there is $r \in [1,t)$ such that both Q_r and Q_{r+1} are flexible and $|Q_r| \leq |Q_{r+1}|$, then $Q_r = Q_{r+1}$ and $\mathcal{A}$ has representation*

$$\mathsf{merge}_r(\rho) = \langle X, (Q_i, \ell_i, u_i)_{i=1}^{r-1} \cdot (Q_r, \ell_r + \ell_{r+1}, u_r + u_{r+1}) \cdot (Q_i, \ell_i, u_i)_{i=r+2}^{t} \rangle.$$

3. *If there is $r \in [1,t]$ such that $\ell_r > 1$, then $\mathcal{A}$ has representation*

$$\mathsf{split}_r(\rho) = \langle X, (Q_i, \ell_i, u_i)_{i=1}^{r-1} \cdot (Q_r, \ell_r - 1, \ell_r - 1) \cdot (Q_r, 1, u_r - \ell_r + 1) \cdot (Q_i, \ell_i, u_i)_{i=r+1}^{t} \rangle.$$

4. *If Q_1 is fixed, then $\mathcal{A}$ has representation* $\mathsf{truncate}(\rho) = \langle X Q_1^{\ell_1}, (Q_{i+1}, \ell_{i+1}, u_{i+1})_{i=1}^{t-1} \rangle$.

Proof. Statements (3) and (4) are trivial. For (2), if $|Q_r| \leq |Q_{r+1}|$ and both Q_r and Q_{r+1} are flexible, then Lemma 3.5 implies $Q_r = Q_{r+1}$. Hence the statement follows.

Finally, we show that statement (1) holds. Assume that Q_r is flexible and Q_{r+1} is fixed. Then Lemma 3.4 implies that $|Q_r|$ is a period of $Q_r Q_{r+1}^{\ell_{r+1}}$, and thus $Q_{r+1}^{\ell_{r+1}} = Q_r^x Q_r[1..y]$ with $x = \lfloor |Q_{r+1}^{\ell_{r+1}}| / |Q_r| \rfloor$ and $y = |Q_{r+1}^{\ell_{r+1}}| \bmod |Q_r|$. (Either x or y might be zero, but this is irrelevant for the proof.) Let $P = Q_r[1..y]$ and $S = Q_r(y..|Q_r|]$. Any rotation of a primitive string is primitive, and hence $\mathsf{rot}^y(Q_r) = SP$ is primitive. For any exponent $a \in [\ell_r, u_r]$, it holds $Q_r^a Q_{r+1}^{\ell_{r+1}} = (PS)^a (PS)^x P = (PS)^x P (SP)^a = Q_{r+1}^{\ell_{r+1}} (\mathsf{rot}^y(Q_r))^a$. Hence the stated transformation does not change the represented affine prefix set. ◀

The leftmost (i.e., lowest index) fixed component Q_r of a representation can either be removed with truncate (if $r = 1$), or it can be moved further to the left with switch_{r-1} (if $r > 1$). By repeatedly applying truncate and switch, we obtain the following lemma.

▶ **Lemma 3.8.** *Let $\rho = \langle X, (Q_i, \ell_i, u_i)_{i=1}^{t} \rangle$ be a representation of an affine prefix set, and let $F = \{j \in [1,t] \mid \ell_j < u_j\} = \{j_1, \ldots, j_{|F|}\}$ with $j_1 < j_2 < \cdots < j_{|F|}$ be the indices of the flexible components. Then the affine prefix set has a representation $\langle \hat{X}, (\hat{Q}_{j_i}, \ell_{j_i}, u_{j_i})_{i=1}^{|F|} \rangle$ such that $\hat{Q}_{j_i}$ is a rotation of Q_{j_i} for every $i \in [1, |F|]$. Both $\hat{X}$ and all the $\hat{Q}_{j_i}$ are functions of X, $Q_1, \ldots, Q_t$, and $\ell_1, \ldots, \ell_t$, i.e., they are independent of $u_1, \ldots, u_t$.*

After applying Lemma 3.8, we repeatedly apply merge to remove all inversions. Then, we apply split until all flexible components have exponent lower bound 1. This may result in new fixed components, which we remove with Lemma 3.8, resulting in the following lemma.

▶ **Lemma 3.9.** *An affine prefix set represented by $\langle X, (Q_i, \ell_i, u_i)_{i=1}^{t} \rangle$ has an irreducible representation of order $|L| \leq t$, where $L = \{|Q_r| \mid r \in [1,t] : \ell_r < u_r\}$.*

3.2 Strongly affine representations

To analyze the intricate structure of repetitive fragments induced by affine prefix sets, it is helpful to assume that periodicity extends slightly beyond the region in question. To this end, we define a *strongly affine representation* of an affine prefix set of T, in which the exponent upper bound of each (flexible) component can be increased by five and still yield an affine prefix set of T. A supplementary drawing is provided in Figure 1b.

▶ **Definition 3.10** (Strongly affine representations). *A representation $\rho = \langle X, (Q_i, \ell_i, u_i)_{i=1}^t \rangle$ of an affine prefix set of a string T is* strongly affine *if and only if its* periodic expansion $\mathsf{expand}(\rho) = \langle X, (Q_i, \ell_i, u'_i)_{i=1}^t \rangle$ *is also the representation of an affine prefix set of T, where u'_i, $i \in [1, t]$, is defined as follows: $u'_i = u_i$ if $u_i = \ell_i$ and $u'_i = u_i + 5$ otherwise.*

▶ **Definition 3.11** (Canonical representation). *A representation of an affine prefix set is* canonical *if and only if it is both strongly affine and irreducible.*

It can be readily verified that, if ρ is strongly affine, then also $\mathsf{truncate}(\rho)$, $\mathsf{split}_r(\rho)$, $\mathsf{merge}_r(\rho)$, and $\mathsf{switch}_r(\rho)$ are strongly affine (for any r, assuming that the respective operation is indeed applicable). We obtained Lemma 3.9 by applying a sequence of these operations, and hence we have the following immediate corollary.

▶ **Corollary 3.12.** *An affine prefix set with strongly affine representation $\langle X, (Q_i, \ell_i, u_i)_{i=1}^t \rangle$ has a canonical representation of order $|L| \leq t$, where $L = \{|Q_r| \mid r \in [1, t] : \ell_r < u_r\}$.*

Whether a representation ρ of an affine prefix set $\mathcal{A}$ of T is strongly affine does not only depend on ρ, it also depends on what T looks like beyond the end of the longest prefix represented by ρ. Therefore, one cannot hope to transform an arbitrary representation into an equivalent strongly affine representation. However, by "removing" the last five copies of each component and treating them separately, we show that we can cover an affine prefix set of order t with at most 6^t canonical representations.

▶ **Lemma 3.13.** *An affine prefix set of order t can be partitioned into at most 6^t affine prefix sets, each of which has a canonical representation of order at most t.*

Proof. Let $\langle X, (Q_i, \ell_i, u_i)_{i=1}^t \rangle$ be a representation of an affine prefix set. We produce a set of representations $R = \{\langle X, (Q_i, \ell'_i, u'_i)_{i=1}^t \rangle \mid \forall r \in [1, t] : (\ell'_r, u'_r) \in B_r\}$, where $\forall r \in [1, t]$, we define $B_r = \{(u, u) \mid u \in [\max(\ell_r, u_r - 4), u_r]\} \cup \{(\ell_r, \max(\ell_r, u_r - 5))\}$. It is easy to see that the affine sets generated by representations in R form a partition of the affine set generated by $\langle X, (Q_i, \ell_i, u_i)_{i=1}^t \rangle$. By design, for any representation in R, and for any component Q_r, we know that Q_r is either fixed, or it has exponent lower bound ℓ_r and exponent upper bound $u_r - 5$. Hence the instances in R are strongly affine, and it follows from Corollary 3.12 that each of them has an equivalent canonical representation of order at most t. Finally, it holds $\forall i \in [1, t] : |B_i| \leq 6$ and thus $|R| = \prod_{i=1}^t |B_i| \leq 6^t$. ◀

By applying the technique from the proof above to the prefix-palindromes, i.e., to each of the representations of order 1 produced by Lemma 3.2, we obtain the following result.

▶ **Corollary 3.14.** *The set of prefix-palindromes of a string $T[1..n]$ can be partitioned into $\mathcal{O}(\log n)$ affine sets of order at most 1. Each set of order 1 has canonical representation $\langle U(VU)^\ell, (VU, 1, u) \rangle$ for some $U \in \mathsf{PAL} \cup \{\varepsilon\}$, $V \in \mathsf{PAL}$ and integers $\ell \geq 1$ and $u > 1$.*

▶ **Corollary 3.15.** *Let $\langle X, (Q_i, \ell_i, u_i)_{i=1}^t \rangle$ be a canonical representation of an affine prefix set. Then it holds $\forall r \in [1, t] : |Q_r| > \sum_{j=r+1}^t |Q_j^{u_j+4}|$.*

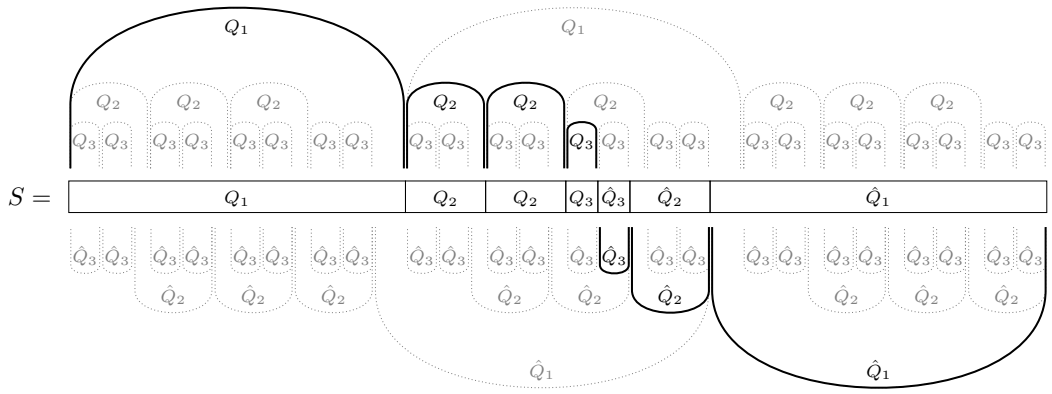


Figure 2 Lemma 3.18 applied to an irreducible representation $\langle X, (Q_1, 1, 2) \cdot (Q_2, 1, 3) \cdot (Q_3, 1, 2)\rangle$. The drawing shows the longest prefix $S = Q_1^2 Q_2^3 Q_3^2$ generated by the representation. By the lemma, for any $a_1 \in [0, 2]$, $a_2 \in [0, 3]$ and $a_3 \in [0, 2]$, it holds $S = Q_1^{2-a_1} Q_2^{3-a_2} Q_3^{2-a_3} \cdot \hat{Q}_3^{a_3} \hat{Q}_2^{a_2} \hat{Q}_1^{a_1}$, where each $\hat{Q}_j$ is the length-$|Q_j|$ suffix of S. The drawing highlights the case where $a_1 = a_2 = a_3 = 1$.

Proof. If $\rho = \langle X, (Q_i, \ell_i, u_i)_{i=1}^t\rangle$ is canonical, then clearly $\mathsf{expand}(\rho) = \langle X, (Q_i, \ell_i, u_i + 5)_{i=1}^t\rangle$ is irreducible. Thus, the statement follows from Lemma 3.5 applied to $\mathsf{expand}(\rho)$. ◀

▶ **Lemma 3.16.** *Let $\langle X, (Q_i, 1, u_i)_{i=1}^t\rangle$ be a canonical representation of an affine prefix set, and let $h \in [0, 5]$. Then $\langle \varepsilon, (Q_i, 1, u_i + h)_{i=2}^t\rangle$ is an irreducible representation of an affine prefix set of the string Q_1^2, and, if $h < 5$, also of the string Q_1.*

Proof. Consider any $h \in [0, 5]$. Due to the strong affinity, $\langle X, (Q_i, 1, u_i + h)_{i=1}^t\rangle$ represents an affine prefix set. Let $(a_i)_{i=2}^t$ be a sequence of exponents with $\forall j \in [2, t] : a_j \in [1, u_i + h]$. By Lemma 3.4, the string $Q_1 Q_2^{a_2} Q_3^{a_3} \cdots Q_t^{a_t}$ has period $|Q_1|$. Due to Lemma 3.5, it holds $|Q_2^{a_2} Q_3^{a_3} \cdots Q_t^{a_t}| < |Q_1 Q_2| < |Q_1^2|$. Hence we have shown that $Q_2^{a_2} Q_3^{a_3} \cdots Q_t^{a_t}$ is a prefix of Q_1^2, and $\langle \varepsilon, (Q_i, 1, u_i + h)_{i=2}^t\rangle$ is a representation of an affine prefix set of Q_1^2. Since $\langle X, (Q_i, 1, u_i)_{i=1}^t\rangle$ is irreducible, it is easy to see that also $\langle \varepsilon, (Q_i, 1, u_i + h)_{i=2}^t\rangle$ is irreducible.

If $h < 5$, then Lemma 3.5 invoked with $\langle X, (Q_i, 1, u_i + 5)_{i=1}^t\rangle$ implies $|Q_2^{a_2} Q_3^{a_3} \cdots Q_t^{a_t}| < |Q_1|$, and $\langle \varepsilon, (Q_i, 1, u_i + h)_{i=2}^t\rangle$ indeed only generates strings of length less than $|Q_1|$. ◀

▶ **Corollary 3.17.** *Let $\langle X, (Q_i, 1, u_i)_{i=1}^t\rangle$ be a canonical representation of an affine prefix set. Then $\langle \varepsilon, (Q_i, 1, u_i)_{i=2}^t\rangle$ is a canonical representation of an affine prefix set of the string Q_1^2.*

Proof. By Lemma 3.16, $\langle \varepsilon, (Q_i, 1, u_i + 5)_{i=2}^t\rangle$ is an irreducible representation of an affine prefix set of Q_1^2. Hence $\langle \varepsilon, (Q_i, 1, u_i)_{i=2}^t\rangle$ is a canonical representation for Q_1^2. ◀

3.3 Reversing the structure of affine prefix sets

We first show that a periodic fragment of T induced by an affine prefix set can be covered by a combination of a forward and a "backward" affine prefix set (see Figure 2):

▶ **Lemma 3.18.** *Let $\langle X, (Q_i, 1, u_i)_{i=1}^t\rangle$ be an irreducible representation of an affine prefix set, let $S = Q_1^{u_1} Q_2^{u_2} \cdots Q_t^{u_t}$, and for $j \in [1, t]$ let $\hat{Q}_j$ be the length-$|Q_j|$ suffix of S. For any sequence $(a_i)_{i=1}^t$ with $\forall j \in [1, t] : a_j \in [0, u_j]$, it holds*

$$S = Q_1^{u_1 - a_1} Q_2^{u_2 - a_2} \cdots Q_t^{u_t - a_t} \cdot \hat{Q}_t^{a_t} \hat{Q}_{t-1}^{a_{t-1}} \cdots \hat{Q}_1^{a_1}.$$

Proof. If $t = 1$, then $S = Q_1^{u_1} = \hat{Q}_1^{u_1} = Q_1^{u_1-a_1}\hat{Q}_1^{a_1}$. Inductively assume that the lemma holds for representations of order $t-1$. Now we show that it holds for representations of order t. If $\langle X, (Q_i, 1, u_i)_{i=1}^t\rangle$ is an irreducible representation of an affine prefix set, then clearly $\langle XQ_1^{u_1}, (Q_i, 1, u_i)_{i=2}^t\rangle$ is an irreducible representation of another affine prefix set. This representation is of order $t-1$, and hence the inductive assumption implies

$$S = Q_1^{u_1} \cdot Q_2^{u_2-a_2}Q_3^{u_3-a_3}\cdots Q_t^{u_t-a_t} \cdot \hat{Q}_t^{a_t}\hat{Q}_{t-1}^{a_{t-1}}\cdots\hat{Q}_2^{a_2}.$$

If $a_1 = 0$, then there is nothing left to do. Hence assume $a_1 > 0$. Since $\langle X, (Q_i, 1, u_i)_{i=1}^t\rangle$ is an irreducible representation, Lemma 3.4 implies that $|Q_1|$ and therefore also $q = a_1 \cdot |Q_1|$ is a period of S. Hence S has a border of length $s-q$, where $s = |S|$, and it holds

$$S[1..s-q] = S[1+q..s] = Q_1^{u_1-a_1} \cdot Q_2^{u_2-a_2}Q_3^{u_3-a_3}\cdots Q_t^{u_t-a_t} \cdot \hat{Q}_t^{a_t}\hat{Q}_{t-1}^{a_{t-1}}\cdots\hat{Q}_2^{a_2}.$$

Finally, as mentioned before, $S[s-q+1..s]$ of length $q = a_1 \cdot |Q_1|$ has period $|Q_1|$. Hence $S[s-q+1..s] = (S[s-|Q_1|+1..s])^{a_1} = \hat{Q}_1^{a_1}$, which concludes the proof. ◀

We now build on this characterization to convert irreducible representations of affine prefix sets of S into irreducible representations of affine prefix sets of $\mathsf{rev}(S)$.

▶ **Corollary 3.19.** *Let $\langle X, (Q_i, 1, u_i)_{i=1}^t\rangle$ be a canonical representation of an affine prefix set, let $s = \sum_{i=2}^t (u_i+1)\cdot|Q_i|$, and for $j \in [1,t]$ let $\hat{Q}_j$ be the length-$|Q_j|$ suffix of $\mathsf{rot}^s(Q_1)$. Then $\langle \varepsilon, (\mathsf{rev}(\hat{Q}_i), 1, u_i)_{i=2}^t\rangle$ represents an affine prefix set of $\mathsf{rev}(\mathsf{rot}^s(Q_1))$.*

Proof. Consider any sequence $(a_i)_{i=2}^t$ of exponents admitted by the representation, i.e., $\forall j \in [2,t] : a_j \in [1, u_j]$. By Lemma 3.16, $\langle \varepsilon, (Q_i, 1, u_i+1)_{i=2}^t\rangle$ is an irreducible representation of an affine prefix set of Q_1, which implies $Q_1[1..s] = Q_2^{u_2+1}Q_3^{u_3+1}\cdots Q_t^{u_t+1}$. For this representation, Lemma 3.18 implies that $\hat{Q}_t^{a_t}\hat{Q}_{t-1}^{a_{t-1}}\cdots\hat{Q}_2^{a_2}$ is a suffix of $Q_1[1..s]$. Thus, its reversal $\mathsf{rev}(\hat{Q}_t^{a_t}\hat{Q}_{t-1}^{a_{t-1}}\cdots\hat{Q}_2^{a_2}) = \mathsf{rev}(\hat{Q}_2^{a_2})\mathsf{rev}(\hat{Q}_3^{a_3})\cdots\mathsf{rev}(\hat{Q}_t^{a_t})$ is a prefix of $\mathsf{rev}(Q_1[1..s])$, which is a prefix of $\mathsf{rev}(\mathsf{rot}^s(Q_1))$. ◀

4 Appending a Palindrome to an Affine Prefix Set

In this section, we show how to extend an affine prefix set $\mathcal{A}$ with a palindrome. This amounts to computing the union of affine prefix sets where each new prefix is formed by concatenating a prefix from $\mathcal{A}$ with a palindrome. We distinguish two cases, depending on whether the appended palindrome lies within a periodic fragment of T. In either case, we may temporarily overextend $\mathcal{A}$, producing sets that are not affine prefix sets. We then restore validity by truncating the sets using the lemma below. For a set of strings $\mathcal{A}$, denote $\mathcal{A}|_m = \{S \in \mathcal{A} : |S| \le m\}$.

▶ **Lemma 4.1.** *Let $\langle X, (Q_i, \ell_i, u_i)_{i=1}^t\rangle$ be a representation of an affine prefix set $\mathcal{A}$. For $m \in \mathbb{N}$, we can express $\mathcal{A}' = \mathcal{A}|_m$ as a union of at most $t' \le t$ affine prefix sets $\mathcal{A}' = \bigcup_{j=1}^{t'} \mathcal{A}_j$, each with a representation of order at most t.*

Proof Sketch. The proof is by induction. For the base case $t = 1$, it is enough to reduce the upper bound on the exponent of Q_1. For $t > 1$, the proof proceeds by finding a minimal exponent $a_1 \ge \ell_1$ such that $|XQ_1^{a_1}Q_2^{u_2}\cdots Q_t^{u_t}| > m$. If $a_1 > u_1$, then $\mathcal{A}' = \mathcal{A}$. Otherwise, we assume w.l.o.g. that $\mathcal{A}$ is irreducible (see Lemma 3.9). It follows that we do not need to consider prefixes generated with an exponent larger than a_1 for Q_1. We partition the remaining prefixes into two sets, one of which can be further broken down using the inductive hypothesis, reducing the problem to problems of smaller sizes until the result follows. ◀

4.1 Appending a long palindrome

Assume that the affine prefix set to be extended is given in a canonical representation $\langle X, (Q_i, 1, u_i)_{i=1}^t \rangle$. We first focus on appending *long palindromes* of length at least $2|Q_1|$, and then we show that the shorter palindromes can be handled recursively. Note that, for a canonical representation, T has a prefix $XQ_1^{u_1+5}$. At the same time, the longest prefix in the affine set is of length less than $XQ_1^{u_1+1}$ by Corollary 3.15. This leads us to a case distinction based on the center of the palindrome to be appended. If the center is before position $|XQ_1^{u_1+3}|$, then we can show that the entire palindrome is within the $|Q_1|$-periodic prefix of $T[|X|+1..n]$. Otherwise, the left half of the palindrome contains position $|XQ_1^{u_1+2}|$, and we can use this position as an anchor point for the extension.

4.1.1 Appending a long palindrome within a run of Q_1

We now focus on the case where the long palindrome to be appended is entirely within the $|Q_1|$-periodic prefix of $T[|X|+1..n]$. We proceed in two steps. First (in Theorem 4.2), we show how to append a palindrome under the assumption that the entire string has the form XQ_1^x for some integer x. Secondly (Corollary 4.3), we truncate the result of the first step so that it corresponds to XQ_1^α, where $\alpha \in \mathbb{Q}$ is the largest value such that XQ_1^α is a prefix of T.

▶ **Theorem 4.2.** *Let $\langle X, (Q_i, 1, u_i)_{i=1}^t \rangle$ be a canonical representation of an affine prefix set $\mathcal{A}$. Let $s = \sum_{i=2}^t (u_i + 1) \cdot |Q_i|$, and for $j \in [1, t]$ let $\hat{Q}_j$ be the length-$|Q_j|$ suffix of $\mathsf{rot}^s(Q_1)$. If $\mathsf{rot}^r(Q_1) = \mathsf{rev}(Q_1)$ for some $r \in [s, s + |Q_1|)$, then*

$$\langle X \cdot Q_1 \cdot Q_1[1..r-s], (\mathsf{rot}^{r-s}(Q_1), 1, x) \cdot (\mathsf{rev}(\hat{Q}_i), 1, u_i)_{i=2}^t \rangle \tag{1}$$

represents an affine prefix set $\mathcal{A}'$ of XQ_1^{x+3}, for any positive integer x. Furthermore:

1. *If $Y' \in \mathcal{A}'$, then there is a string $Y \in \mathcal{A}$ and a palindrome P such that $Y' = YP$.*
2. *For $Y \in \mathcal{A}$ and $P \in \mathsf{PAL}$, if $|P| \geq 2|Q_1|$ and YP is a prefix of XQ_1^{x+1}, then $YP \in \mathcal{A}'$.*

Proof Sketch. The keystone of the proof is Corollary 3.19 which implies that a string $Q' = \mathsf{rev}(\hat{Q}_2)^{a_2}\mathsf{rev}(\hat{Q}_3)^{a_3} \cdots \mathsf{rev}(\hat{Q}_t)^{a_t}$, where $\forall j \in [2, t] : a_j \in [1, u_j]$, is a prefix of

$$\mathsf{rev}(\mathsf{rot}^s(Q_1)) = \mathsf{rot}^{-s}(\mathsf{rev}(Q_1)) = \mathsf{rot}^{-s}(\mathsf{rot}^r(Q_1)) = \mathsf{rot}^{r-s}(Q_1).$$

Using this fact, we first establish that a string XS' generated by the canonical representation in Equation (1) is a prefix of XQ_1^{x+3}. Next, we show that S' can be decomposed as SP, where $XS \in \mathcal{A}$ and P is a palindrome. It follows that P is a substring of a power of Q_1, and the condition $\mathsf{rot}^r(Q_1) = \mathsf{rev}(Q_1)$ ensures P is a palindrome. Finally, we conclude by considering any string $S \in \mathcal{A}$ and a sufficiently long palindrome P such that SP is a prefix of Q_1^{x+1}. Due to $XS \in \mathcal{A}$, there is some sequence $\forall j \in [1, t] : a_j \in [1, u_j]$ of exponents such that $S = Q_1^{u_1-a_1+1}Q_2^{u_2-a_2+1} \cdots Q_t^{u_t-a_t+1}$. From this and the fact that P is a palindrome, we show that XSP fits the structure required for membership in $\mathcal{A}'$, completing the proof. ◀

By combining Theorem 4.2 and Lemma 4.1, we obtain:

▶ **Corollary 4.3.** *Let $\langle X, (Q_i, 1, u_i)_{i=1}^t \rangle$ be a canonical representation of an affine prefix set $\mathcal{A}$. Let $\alpha \in \mathbb{Q}$ be the largest possibly fractional exponent such that XQ_1^α is a prefix of T, and define $\mathcal{S} = \{S \cdot P : S \cdot P \text{ is a prefix of } XQ_1^\alpha, S \in \mathcal{A}, P \in \mathsf{PAL}, |P| \geq 2|Q_1|\}$. There are $t' \leq t$ affine prefix sets $\mathcal{B}_i$, $i \in [1, t']$, each of order $\leq t$, such that for $\mathcal{B} = \bigcup_{i=1}^{t'} \mathcal{B}_i$ we have $\mathcal{S} \subseteq \mathcal{B}$ and for every $Y' \in \mathcal{B}$, there is a string $Y \in \mathcal{A}$ and a palindrome P such that $Y' = YP$.*

4.1.2 Appending a long palindrome outside a run of Q_1

▶ **Theorem 4.4.** *Let $\langle X, (Q_i, 1, u_i)_{i=1}^t\rangle$ be a canonical representation of an affine prefix set $\mathcal{A}$ and $s = \sum_{i=2}^t (u_i+1)\cdot|Q_i|$. For $j \in [1,t]$, let $\hat{Q}_j$ be the length-$|Q_j|$ suffix of $\mathsf{rot}^s(Q_1)$. For any string P, $\langle X \cdot Q_1^{u_1+2} \cdot P \cdot \mathsf{rev}(Q_1)[1..|Q_1|-s], (\mathsf{rev}(\hat{Q}_i), 1, u_i)_{i=1}^t\rangle$ represents an affine prefix set $\mathcal{A}'$ of the string $X \cdot Q_1^{u_1+2} \cdot P \cdot \mathsf{rev}(Q_1^{u_1+2})$, where $\mathcal{A}' = \{SWP \cdot \mathsf{rev}(W) \mid S \in \mathcal{A}$ and $SW = X \cdot Q_1^{u_1+2}\}$.*

Proof Sketch. Let $q = |Q_1|$. We can split the output representation into a concatenation

$$\langle X \cdot Q_1^{u_1+2} \cdot P \cdot \mathsf{rev}(Q_1)[1..q-s], (\mathsf{rev}(\hat{Q}_1), 1, u_1)\rangle \cdot \langle \varepsilon, (\mathsf{rev}(\hat{Q}_i), 1, u_i)_{i=2}^t\rangle. \tag{2}$$

We first apply Corollary 3.19 to deduce that Equation (2) represents an affine prefix set of the string $X\cdot Q_1^{u_1+2}\cdot P\cdot\mathsf{rev}(Q_1^{u_1+2})$. Secondly, we show that every element in $\mathcal{A}$ contributes exactly one element to $\mathcal{A}'$, and hence $|\mathcal{A}'| = |\mathcal{A}|$. It thus suffices to show that any string generated by Equation (2) is in $\mathcal{A}'$. It then readily follows that Equation (2) generates exactly $\mathcal{A}'$. To do so, we consider any string S' generated by Equation (2). Such a string must be of the form $S' = XQ_1^{u_1+2}P\cdot\mathsf{rev}(W)$, where $\mathsf{rev}(W) = \mathsf{rev}(Q_1)[1..q-s]\cdot\mathsf{rev}(\hat{Q}_1)^{a_1}\mathsf{rev}(\hat{Q}_2)^{a_2}\cdots\mathsf{rev}(\hat{Q}_t)^{a_t}$ for some exponents $\forall i \in [1,t] : a_i \in [1,u_i]$. By our previous observations, $\mathsf{rev}(W)$ is a prefix of $\mathsf{rev}(Q_1)^{u_1+2}$, and thus there is a unique string S such that $SW = XQ_1^{u_1+2}$ and $S' = SWP\cdot\mathsf{rev}(W)$. It remains to be shown that $S \in \mathcal{A}$, which then implies $S' \in \mathcal{A}'$. For this purpose, we carefully analyze the length of S and show that a prefix of length $|S|$ indeed belongs to $\mathcal{A}$, concluding the proof. ◀

For a fragment $P = T[x..y]$ of T, denote its center $(x+y)/2$ by $\mathrm{cen}(P)$.

▶ **Corollary 4.5.** *Let $\langle X, (Q_i, 1, u_i)_{i=1}^t\rangle$ be a canonical representation of an affine prefix set $\mathcal{A}$, and consider the set of strings $\mathcal{A}' = \{S\cdot P : S\cdot P$ is a prefix of $T, S \in \mathcal{A}, P \in \mathsf{PAL}, \mathrm{cen}(P) > |XQ_1^{u_1+3}|\}$. There are $t' = \mathcal{O}(t\log n)$ affine prefix sets $\mathcal{B}_i$, $i \in [1,t']$, each of order $\le t+1$, such that both of the following properties hold for $\mathcal{B} = \bigcup_{i=1}^{t'}\mathcal{B}_i$:*

1. *$\mathcal{A}' \subseteq \mathcal{B}$.*
2. *For every $Y' \in \mathcal{B}$, there is a string $Y \in \mathcal{A}$ and a palindrome P such that $Y' = YP$.*

Proof Sketch. Consider any $SP \in \mathcal{A}'$, where SP is a prefix of T, $S \in \mathcal{A}$, $P \in \mathsf{PAL}$, $\mathrm{cen}(P) > |XQ_1^{u_1+3}|$. Due to $S \in \mathcal{A}$, Corollary 3.15 implies $|S| < |XQ_1^{u_1+1}|$. Let $P' = T[x..y]$, where $x = 1 + |XQ_1^{u_1+2}|$ and $y = 2\cdot\mathrm{cen}(P) - x$. We claim that $P' \in \mathsf{PAL}$. Indeed, the starting position $|S|+1$ of P is less than the starting position x of P', and the centers of P and P' coincide with $\mathrm{cen}(P) - x = y - \mathrm{cen}(P)$. We call P' the *core palindrome* of SP. Note that every core palindrome is a prefix of $T[x..n]$ (which is independent of SP). Therefore, by Corollary 3.14, the set of core palindromes can be represented as the union of $\mathcal{O}(\log n)$ affine prefix sets. Let $\mathcal{C}$ be any of these sets. We now describe how to compute the part of $\mathcal{A}'$ that contains strings of the form $SP = SWP'\cdot\mathsf{rev}(W)$, where $S \in \mathcal{A}$, $P \in \mathsf{PAL}$, and the core palindrome of SP is some $P' \in \mathcal{C}$. The procedure depends on the representation of $\mathcal{C}$, which, by Corollary 3.14, is covered by one of the following cases. Let $q = |Q_1|$.

Case 1: $\mathcal{C}$ is given in strongly affine representation $\langle U\cdot(VU)^\ell, (VU, 1, u)\rangle$, where VU is primitive and $|VU| > q$. In this case, we consider one fixed core palindrome in $\mathcal{C}$ and apply Theorem 4.4 and Lemma 4.1 to obtain an affine prefix set generated by it. We then show that the sets generated by other core palindromes have a similar representation, which allows to union them and to obtain the final affine prefix set.

Case 2: $\mathcal{C}$ is given in representation $\langle P', \varepsilon\rangle$ of order 0, i.e., it contains a single core palindrome P'. We proceed exactly like in Case 1, but with a single palindrome.

Case 3: $\mathcal{C}$ has strongly affine representation $\langle U \cdot (VU)^\ell, (VU, 1, u)\rangle$, where VU is primitive and $|VU| = q$. For $i \in [1, u]$, let $P_i = U \cdot (VU)^{\ell+i}$. We show that, if $\mathcal{A}'$ contains some $SP = S \cdot W \cdot P_i \cdot \mathsf{rev}(W) = XQ_1^{u_1+2}P_i \cdot \mathsf{rev}(W)$ with $S \in \mathcal{A}$, then the entire SP can be written as XQ_1^α for some $\alpha \in \mathbb{Q}$. Hence we can simply apply Corollary 4.3 and obtain that the affine prefix set generated by $\mathcal{C}$ is the union of at most t affine prefix sets of order at most t.

Case 4: $\mathcal{C}$ has strongly affine representation $\langle U \cdot (VU)^\ell, (VU, 1, u)\rangle$, where VU is primitive and $|VU| < q$. We show that this case is impossible due to primitivity of Q_1.

We call the created affine prefix sets $\mathcal{B}_i$. There are $\mathcal{O}(\log n)$ core palindrome sets, each handled by a single case. Each case creates $\leq t$ representations of order $\leq t+1$. Hence, there are $\mathcal{O}(t \log n)$ affine prefix sets $\mathcal{B}_i$ in total, each of order $\leq t+1$ Furthermore, the four cases cover all possibilities, and hence $\mathcal{A}' \subseteq \mathcal{B} = \bigcup_{i=1}^{t'} \mathcal{B}_i$. The second property holds by construction of the sets $\mathcal{B}_i$. ◀

4.1.3 Appending all long palindromes and recursion

▶ **Lemma 4.6.** *Let $\langle X, (Q_i, 1, u_i)_{i=1}^t\rangle$ be a canonical representation of an affine prefix set $\mathcal{A}$ and $\mathcal{A}' = \{S \cdot P \mid S \in \mathcal{A}, P \in \mathsf{PAL}, |P| \geq 2|Q_1|$, and $S \cdot P$ is a prefix of $T\}$. There are $t' = \mathcal{O}(t \log n)$ affine prefix sets $\mathcal{B}_i$, $1 \leq i \leq t'$, each of order $\leq t+1$, such that $\mathcal{A}' \subseteq \cup_{i=1}^{t'}\mathcal{B}_i$ and for each string $S' \in \cup_{i=1}^{t'}\mathcal{B}_i$, there is a string $S \in \mathcal{A}$ and $P \in \mathsf{PAL}$ such that $S' = S \cdot P$.*

Proof. We consider the sets from Corollaries 4.3 and 4.5, defined by

$$\mathcal{A}_1 = \{S \cdot P : S \cdot P \text{ is a prefix of } XQ_1^\alpha, S \in \mathcal{A}, P \in \mathsf{PAL}, |P| \geq 2|Q_1|\} \text{ and}$$

$$\mathcal{A}_2 = \{S \cdot P : S \cdot P \text{ is a prefix of } T, S \in \mathcal{A}, P \in \mathsf{PAL}, \mathrm{cen}(P) > |X| + (u_1+3) \cdot |Q_1|\},$$

where α is the largest (possibly fractional) exponent such that XQ_1^α is a prefix of T. Due to Corollaries 4.3 and 4.5, we can express (a superset of) $\mathcal{A}_1 \cup \mathcal{A}_2$ as the union of $\mathcal{O}(t \log n)$ affine prefix sets, each of order $\leq t+1$, where every string in each of the prefix sets is the concatenation of a string from $\mathcal{A}$ and a palindrome. It remains to be shown that $\mathcal{A}' \subseteq \mathcal{A}_1 \cup \mathcal{A}_2$. For the sake of contradiction, assume that there is some string $SP \in \mathcal{A}' \setminus (\mathcal{A}_1 \cup \mathcal{A}_2)$, where $S \in \mathcal{A}$, $P \in \mathsf{PAL}$ and $|P| \geq 2|Q_1|$. Due to $SP \notin \mathcal{A}_1$, SP is not a prefix of XQ_1^α and hence it must be longer than XQ_1^α. Let $m = |XQ_1^\alpha| - |S|$. We show a lower bound on m. Since the given representation is strongly affine, it holds $\alpha \geq u_1 + 5$. It is also irreducible, and hence Corollary 3.15 implies $|S| < |XQ_1^{u_1+1}|$. Therefore, it holds $m > 4|Q_1|$. Note that P does not have period $|Q_1|$, but its length-m prefix, which is a suffix of Q_1^α, does. Hence, by Lemma 2.3, it follows that P is of length over $2m - |Q_1|$, and therefore

$$\mathrm{cen}(P) \geq |S| + |P|/2 > |S| + m - |Q_1|/2 = |XQ_1^\alpha| - |Q_1|/2 > |XQ_1^{u_1+4}|.$$

This implies $SP \in \mathcal{A}_2$, which contradicts the initial assumption. ◀

We have shown that appending palindromes of length at least $2|Q_1|$ results in $\mathcal{O}(t \log n)$ affine prefix sets of order $\leq t+1$. For appending shorter palindromes, we exploit properties of strongly affine prefix sets that allow us to apply the previously described approach recursively.

▶ **Lemma 4.7.** *Let $\langle X, (Q_i, 1, u_i)_{i=1}^t\rangle$ be a canonical representation of an affine prefix set $\mathcal{A}$ and $\mathcal{A}' = \{S \cdot P : S \cdot P$ is a prefix of $T, S \in \mathcal{A}, P \in \mathsf{PAL}\}$ a set of strings. Then $\mathcal{A}'$ is a union of $\mathcal{O}((t+1)^2 \log n)$ affine prefix sets, each of order $\leq t+1$.*

▶ **Theorem 1.1.** *Let $0 < \epsilon < 1$ be constant, $T[1..n]$ a string, and $k \in \mathbb{N}^+$. The set of prefixes of T that belong to PAL^k is the union of $\mathcal{O}(6^{k^2/(2-\epsilon)} \cdot \log^k n)$ affine prefix sets of order $\leq k$.*

Proof. We start with the empty affine prefix set representing PAL^0. We proceed in k levels $k' \in [0, k)$. The union of the affine prefix sets of level k' is exactly the set of all k'-palindromic prefixes of T. For each affine prefix set of the current level k', we first apply Lemmas 3.9 and 3.13 to obtain at most $6^{k'}$ canonical representations of order $\leq k'$. Then, for each of the representations, we append a palindrome using Lemma 4.7, resulting in at most $c \cdot (k'+1)^2 \log n$ affine prefix sets of order at most $k'+1$, which we move to level $k'+1$. Here, c is a positive constant that depends on the precise complexity analysis of Lemma 4.7. Hence, after processing level $k-1$, the total number of affine prefix sets is bounded by $\prod_{k'=0}^{k-1}(6^{k'} \cdot c \cdot (k'+1)^2 \log n) \leq (k!)^2 \cdot c^k \cdot 6^{(k^2/2)} \cdot \log^k n$. For all sufficiently large k (depending only on ϵ and c), the bound simplifies to $6^{k^2/(2-\epsilon)} \cdot \log^k n$. (And for the remaining values of k, we have $(k!)^2 \cdot c^k \cdot 6^{(k^2/2)} = \mathcal{O}(1)$.) ◀

▶ Remark 4.8. Lemmas 3.9 and 3.13 only work with the lengths of the components in the representations of affine prefix sets and the exponents bounds. Since all affine prefix sets are of small order, it is not difficult to see that these lemmas can be implemented efficiently. To implement Lemma 4.7, we make use of two procedures: The first one computes the longest prefix of a string of form XQ^α, where $\langle X, (Q_i, 1, u_i)_{i=1}^t \rangle$ is a representation of one of the sets, and the second computes a representation of the set of prefix-palindromes of a string. In the read-only model, both procedures can be implemented in $\mathcal{O}(n)$ time and $\mathcal{O}(\log n)$ space. Bounding the number of calls by the number of the generated affine prefix sets, we finally obtain Theorem 1.3. The algorithm of Theorem 1.3 can then be used to test if the palindromic length of T is at most k by checking whether T is a k-palindromic prefix, however, to achieve even better complexity, we run two copies of the algorithm, one from the left and the other from the right, and then combine their results to obtain Theorem 1.4.

References

1. Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. If the current clique algorithms are optimal, so is Valiant's parser. *SIAM J. Comput.*, 47(6):2527–2555, 2018. doi:10.1137/16M1061771.
2. Rajeev Alur and P. Madhusudan. Visibly pushdown languages. In *STOC*, pages 202–211, 2004. doi:10.1145/1007352.1007390.
3. Amihood Amir and Benny Porat. Approximate on-line palindrome recognition, and applications. In *CPM*, pages 21–29, 2014. doi:10.1007/978-3-319-07566-2_3.
4. Ajesh Babu, Nutan Limaye, Jaikumar Radhakrishnan, and Girish Varma. Streaming algorithms for language recognition problems. *Theor. Comput. Sci.*, 494:13–23, 2013. doi:10.1016/J.TCS.2012.12.028.
5. Gabriel Bathie, Jonas Ellert, and Tatiana Starikovskaya. Small space encoding and recognition of k-palindromic prefixes. *CoRR*, abs/2410.03309, 2024. doi:10.48550/arXiv.2410.03309.
6. Gabriel Bathie, Tomasz Kociumaka, and Tatiana Starikovskaya. Small-space algorithms for the online language distance problem for palindromes and squares. In *ISAAC*, pages 10:1–10:17, 2023. doi:10.4230/LIPICS.ISAAC.2023.10.
7. Gabriel Bathie and Tatiana Starikovskaya. Property testing of regular languages with applications to streaming property testing of visibly pushdown languages. In *ICALP*, pages 119:1–119:17, 2021. doi:10.4230/LIPICS.ICALP.2021.119.
8. Petra Berenbrink, Funda Ergün, Frederik Mallmann-Trenn, and Erfan Sadeqi Azer. Palindrome recognition in the streaming model. In *STACS*, pages 149–161, 2014. doi:10.4230/LIPICS.STACS.2014.149.

9 Allan Borodin and Stephen A. Cook. A time-space tradeoff for sorting on a general sequential model of computation. In *STOC*, pages 294–301, 1980. doi:10.1145/800141.804677.

10 Kirill Borozdin, Dmitry Kosolobov, Mikhail Rubinchik, and Arseny M. Shur. Palindromic length in linear time. In *CPM*, pages 23:1–23:12, 2017. doi:10.4230/LIPIcs.CPM.2017.23.

11 Bartlomiej Dudek, Pawel Gawrychowski, Garance Gourdel, and Tatiana Starikovskaya. Streaming regular expression membership and pattern matching. In *SODA*, pages 670–694, 2022. doi:10.1137/1.9781611977073.30.

12 Gabriele Fici, Travis Gagie, Juha Kärkkäinen, and Dominik Kempa. A subquadratic algorithm for minimum palindromic factorization. *J. Discrete Algorithms*, 28:41–48, 2014. doi:10.1016/J.JDA.2014.08.001.

13 Nathan Fine and Herbert Wilf. Uniqueness theorems for periodic functions. *Proceedings of the American Mathematical Society*, 16(1):109–114, 1965. doi:10.2307/2034009.

14 Nathanaël François, Frédéric Magniez, Michel de Rougemont, and Olivier Serre. Streaming property testing of visibly pushdown languages. In *ESA*, pages 43:1–43:17, 2016. doi:10.4230/LIPICS.ESA.2016.43.

15 Michael L. Fredman and Dan E. Willard. BLASTING through the information theoretic barrier with FUSION TREES. In *STOC*, pages 1–7, 1990. doi:10.1145/100216.100217.

16 Zvi Galil. On converting on-line algorithms into real-time and on real-time algorithms for string-matching and palindrome recognition. *SIGACT News*, 7(4):26–30, 1975. doi:10.1145/990502.990505.

17 Zvi Galil and Joel I. Seiferas. A linear-time on-line recognition algorithm for "palstar". *J. ACM*, 25(1):102–111, 1978. doi:10.1145/322047.322056.

18 Moses Ganardi. Visibly pushdown languages over sliding windows. In *STACS*, pages 29:1–29:17, 2019. doi:10.4230/LIPICS.STACS.2019.29.

19 Moses Ganardi, Danny Hucke, Daniel König, Markus Lohrey, and Konstantinos Mamouras. Automata theory on sliding windows. In *STACS*, pages 31:1–31:14, 2018. doi:10.4230/LIPICS.STACS.2018.31.

20 Moses Ganardi, Danny Hucke, and Markus Lohrey. Querying regular languages over sliding windows. In *FSTTCS*, pages 18:1–18:14, 2016. doi:10.4230/LIPICS.FSTTCS.2016.18.

21 Moses Ganardi, Danny Hucke, and Markus Lohrey. Randomized sliding window algorithms for regular languages. In *ICALP*, pages 127:1–127:13, 2018. doi:10.4230/LIPICS.ICALP.2018.127.

22 Moses Ganardi, Danny Hucke, Markus Lohrey, and Tatiana Starikovskaya. Sliding window property testing for regular languages. In *ISAAC*, pages 6:1–6:13, 2019. doi:10.4230/LIPICS.ISAAC.2019.6.

23 Moses Ganardi, Artur Jez, and Markus Lohrey. Sliding windows over context-free languages. In *MFCS*, pages 15:1–15:15, 2018. doi:10.4230/LIPICS.MFCS.2018.15.

24 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.

25 Tomohiro I, Shiho Sugimoto, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Computing palindromic factorizations and palindromic covers on-line. In *CPM*, pages 150–161, 2014. doi:10.1007/978-3-319-07566-2_16.

26 Rahul Jain and Ashwin Nayak. The space complexity of recognizing well-parenthesized expressions in the streaming model: The index function revisited. *IEEE Trans. Inf. Theory*, 60(10):6646–6668, 2014. doi:10.1109/TIT.2014.2339859.

27 Donald E. Knuth, James H. Morris Jr., and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6(2):323–350, 1977. doi:10.1137/0206024.

28 Tomasz Kociumaka, Tatiana Starikovskaya, and Hjalte Wedel Vildhøj. Sublinear space algorithms for the longest common substring problem. In *ESA*, pages 605–617, 2014. doi:10.1007/978-3-662-44777-2_50.

29 Dmitry Kosolobov, Mikhail Rubinchik, and Arseny M. Shur. Pal^k is linear recognizable online. In *SOFSEM*, pages 289–301, 2015. doi:10.1007/978-3-662-46078-8_24.

30 Andreas Krebs, Nutan Limaye, and Srikanth Srinivasan. Streaming algorithms for recognizing nearly well-parenthesized expressions. In *MFCS*, pages 412–423, 2011. doi:10.1007/978-3-642-22993-0_38.

31 Frédéric Magniez, Claire Mathieu, and Ashwin Nayak. Recognizing well-parenthesized expressions in the streaming model. *SIAM J. Comput.*, 43(6):1880–1905, 2014. doi:10.1137/130926122.

32 Glenn K. Manacher. A new linear-time "on-line" algorithm for finding the smallest initial palindrome of a string. *J. ACM*, 22(3):346–351, 1975. doi:10.1145/321892.321896.

33 Mikhail Rubinchik and Arseny M. Shur. EERTREE: an efficient data structure for processing palindromes in strings. *Eur. J. Comb.*, 68:249–265, 2018. doi:10.1016/J.EJC.2017.07.021.

34 Mikhail Rubinchik and Arseny M. Shur. Palindromic k-factorization in pure linear time. In *MFCS*, pages 81:1–81:14, 2020. doi:10.4230/LIPICS.MFCS.2020.81.

35 Anatol O. Slisenko. A simplified proof of the real-time recognizability of palindromes on Turing machines. *J. Sov. Math.*, 15:68–77, 1981. doi:10.1007/BF01404109.

New Algorithmic Directions in Optimal Transport and Applications for Product Spaces

Salman Beigi
Institute for Research in Fundamental Sciences (IPM), Tehran, Iran
TEIAS, Khatam University, Tehran, Iran

Omid Etesami
EPFL, Campus Biotech, Geneva, Switzerland
Institute for Research in Fundamental Sciences (IPM), Tehran, Iran

Mohammad Mahmoody
Charlottesville, VA, USA

Amir Najafi
Sharif University of Technology, Tehran, Iran

Abstract

We consider the problem of optimal transport between two high-dimensional distributions μ, ν in $\mathbb{R}^n$ from a new algorithmic perspective, in which we are given a sample $x \sim \mu$ and we have to find a close $y \sim \nu$ while running in poly(n) time, where n is the size/dimension of x, y. In other words, we are interested in making the running time bounded in dimension of the spaces rather than bounded in the total size of the representations of the two distributions. Our main result is a general algorithmic transport result between any product distribution μ and an arbitrary distribution ν of total cost $\Delta + \delta$ under ℓ_p^p cost; here Δ is the cost of the so-called Knothe–Rosenblatt transport from μ to ν, while δ is a computational error that goes to zero for larger running time in the transport algorithm. For this result, we need ν to be "sequentially samplable" with a "bounded average sampling cost" which is a novel but natural notion of independent interest. In addition, we prove the following.

- We prove an algorithmic version of the celebrated Talagrand's inequality for transporting the standard Gaussian distribution Φ^n to an arbitrary ν under the Euclidean-squared cost. When ν is Φ^n conditioned on a set $\mathcal{S}$ of measure ε, we show how to implement the needed sequential sampler for ν in expected time poly(n/ε), using membership oracle access to $\mathcal{S}$. Hence, we obtain an algorithmic transport that maps Φ^n to $\Phi^n|\mathcal{S}$ in time poly(n/ε) and expected Euclidean-squared distance $O(\log 1/\varepsilon)$, which is optimal for a general set $\mathcal{S}$ of measure ε.
- As corollary, we find the first *computational* concentration (Etesami et al. SODA 2020) result for the Gaussian measure under the Euclidean distance with a *dimension-independent* transportation cost, resolving a question of Etesami et al. More precisely, for any set $\mathcal{S}$ of Gaussian measure ε, we map most of Φ^n samples to $\mathcal{S}$ with Euclidean distance $O\left(\sqrt{\log 1/\varepsilon}\right)$ in time poly(n/ε).

2012 ACM Subject Classification Theory of computation → Algorithm design techniques; Theory of computation → Online algorithms; Mathematics of computing → Probabilistic algorithms; Theory of computation → Probabilistic computation

Keywords and phrases Optimal transport, Randomized algorithms, Concentration bounds

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.10

Related Version *Full Version*: `https://arxiv.org/abs/2509.21502`

Funding *Omid Etesami*: Thanks to Sabanci University and TEIAS for their support during part of this work.

1 Introduction

Optimal transport (OT) is a fundamental problem that arises in mathematics, science, and engineering, including differential geometry [17], transportation planning [40], economics [21], machine learning [34, 38], image registration [23], and seismic tomography [35]. In machine

© Salman Beigi, Omid Etesami, Mohammad Mahmoody, and Amir Najafi;
licensed under Creative Commons License CC-BY 4.0

learning, it has been used in unsupervised learning [46], as a measure of the cost of misclassification [20], to define the fairness of algorithms [11], in Wasserstein GANs [2], for transfer learning [14], and in diffusion generative models [47, 26].

In the optimal transport problem, we would like to transport samples from a source distribution μ to points in the target distribution ν with a minimum expected "transportation cost" $\mathsf{c}(x, y)$ of transporting $x \sim \mu$ to $y \sim \nu$. The study of this problem dates back to the work of Monge [33], who wanted the transportation mapping $A(x) = y$ to be deterministic. Kantorovich [25] reformulated the problem by allowing $A(x)$ to be a randomized (stochastic) mapping. In other words, we now look for a coupling π over the distributions μ, ν with minimum expected transportation cost $\mathbb{E}\,\mathsf{c}(x, y)$, using which we define the optimal cost of transporting μ to ν,

$$\mathsf{T}(\mu, \nu) = \min_{\pi \in \mathcal{C}} \mathop{\mathbb{E}}_{(x,y)\sim\pi} \mathsf{c}(x, y)$$

where $\mathcal{C}$ is the set of all couplings between μ, ν. OT is closely related to the notion of "Wasserstein metric" that generalizes OT using a parameter $p \geq 1$ and is the same for $p = 1$.

As a prominent example of the use of OT in mathematics, Talagrand [43] gave a bound on the optimal transport, under the ℓ_2^2 cost, of the n-dimensional Gaussian measure Φ^n to an arbitrary distribution ν based on the KL-divergence of ν from Φ^n. Using this, he derived an essentially optimal concentration of measure result, showing that for any set $\mathcal{S}$ of measure ε in Φ^n, almost all of the measure Φ^n is within ℓ_2^2 (minimum) distance $O(\ln 1/\varepsilon)$ from $\mathcal{S}$.

Computational OT. Computational aspects of OT have recently become extremely important on their own [38]. In the most common formulation of "computational OT", we would like to compute or estimate $\mathsf{T}(\mu, \nu)$ efficiently. Computing $\mathsf{T}(\mu, \nu)$ is a key tool, e.g., for applications that use OT to quantify a loss that allows one to know "how far" we are from a target goal [4, 6, 7]. A common approach to computing $\mathsf{T}(\mu, \nu)$ is to work with empirical sample sets $\mathcal{S} \sim \mu^m, \mathcal{T} \sim \nu^m$, and find the best OT between the empirical distributions $U_{\mathcal{S}}, U_{\mathcal{T}}$ that are uniform over $\mathcal{S}, \mathcal{T}$ (e.g., see [24, 32] and the references therein). This approximation converges to the quantity $\mathsf{T}(\mu, \nu)$ in the limit, and the OT between $U_{\mathcal{S}}, U_{\mathcal{T}}$ can be computed using the Hungarian algorithm for minimum weighted matching [29]. The popular iterative Sinkhorn algorithm solves a regularized version of the OT problem [42] but it also works with empirical sample sets, that is, i.i.d. samples from the distributions. Using empirical samples, one does not rapidly converge to the optimal OT in some elementary cases. For example, to transport the uniform distribution on the n-dimensional unit cube to itself, the OT between two poly(n)-size empirical versions of the original distribution is $\Theta(\sqrt{n})$ in ℓ_2^2 distance even though the actual OT cost is zero.

Statistical OT. The above approach of using empirical samples $\mathcal{S} \sim \mu^m, \mathcal{T} \sim \nu^m$ can in fact be used to approximate the *transport map* itself from μ to ν, as in [24, 32]. For example, Brenier's theorem [10, 28] asserts that under the ℓ_2^2 cost and suitable conditions, a unique Monge mapping achieves optimal transport, and one can aim at approximating this deterministic mapping. This approach is sometimes known as *statistical* optimal transport [13]. However, this approach needs exponential in n samples for n-dimensional distributions to achieve good approximate results. Some previous works like [24, 32] make improvements on this analysis by assuming further smoothness and structural conditions on the distributions but the curse of dimensionality basically remains intact. More importantly, to the best of our knowledge, no previous work models the algorithmic aspect of searching for the transport map by limiting its algorithm to run in polynomial time over the size of the input $x \sim \mu$.

1.1 Our Contributions

In a nutshell, our contributions are (1) formalizing a new theory of algorithmic transport, (2) obtaining initial results on algorithmic transport for the high-dimensional setting, and (3) obtaining applications for algorithmic transport, e.g., to algorithmic concentration of measure. Each of the items above has multiple aspects that are elaborated in the following.

Algorithmic Transport in Polynomial Time. The common computational OT formulation aims to compute or approximate the optimal transportation cost $\mathsf{T}(\mu, \nu)$, yet it does not answer the key question of how to algorithmically compute the *transport map* efficiently over the size of the given input sample. I.e., suppose that we are given a particular sample $x \sim \mu$ as input, and we would like to map it to $y \sim \nu$ as follows: (1) The mapping shall be computed in polynomial time over the size of the input $|x| = n$. (2) We would like to control the expected cost of the transportation. To point out the subtle distinction between our new algorithmic formulation and the traditional computational OT, in this work we use the term *algorithmic transport* to refer to the task of computing a (randomized) mapping A efficiently based on its input size $|x|$ (e.g., the dimension of x), such that $A(x) \sim \nu$, whenever $x \sim \mu$.

Algorithmic transport, when done optimally, can be used to approximate OT cost efficiently as well. In particular, when the transportation cost is bounded by a constant, using $k = \Theta(\varepsilon^{-2})$ independent samples $(x_1, y_1), \ldots, (x_k, y_k) \sim (x, A(x))^k$, the average $\mathbb{E}_i\, \mathsf{c}(x_i, y_i)$ gives an ε-approximation of the OT, with high probability. However, it is not clear how to do the reverse and obtain algorithmic transport from computational OT.

When μ, ν are one dimensional, for natural (convex) costs such as $\mathsf{c}(x, y) = |x - y|^p, p \geq 1$ one can find simple algorithms that simply use a "monotone" transportation plan [45]. Furthermore, when the distributions μ, ν have small domains of size k, one can use algorithms based on min-cost flows to find a full description of the OT from μ to ν in $\mathrm{poly}(k)$ time [37]. However, our focus is on the high-dimensional setting and finding $\mathrm{poly}(n)$-time computable mappings between distributions of dimension n with *super*-polynomial support. We ask:

> *If μ, ν are n-dimensional distributions, how can we find a $\mathrm{poly}(n)$-time computable transport map from $x \sim \mu$ to $y \sim \nu$ of a small/optimal cost?*

Formalizing and answering the question above in various contexts is the focus of our work. Our studies also bear similarities to the line of work on approximating the total variation distance [5, 16] as it coincides with OT under the Hamming distance.

Transport in High-Dimensional Setting. In this work, we approach the main question above through a study of so-called *causal* transports [30, 3] in high dimension, in which the transporting algorithm A produces $y = (y_1, \ldots, y_n)$ from $x = (x_1, \ldots, x_n)$ in an *online* manner: The algorithm A shall output y_i based on $x_{[i]} = (x_1, \ldots, x_i)$ and before receiving x_{i+1}. Hence we also refer to those transports simply as online. The so-called Knothe-Rosenblatt transport (KR transport for short) [27, 39] is an important online transport with two properties: (1) its reverse is also online, and (2) it follows a "greedy" approach in each round by using a monotone mapping of dimension one. Our motivations for studying online transports is twofold: (1) Despite being a *restriction* on how the transport is done, the "online lens" guides us towards algorithm development; (2) In our eyes, information-theoretic study of online algorithms is interesting. In particular, in Section 2.1, we prove that the KR transport is optimal among all online transports when the source distribution is a product.

Main Result: Algorithmic Transport from Product Distributions. Our main result (Theorem 28) is to design a poly(n)-time online algorithm that transports a generic product distribution $\mu = \mu_1 \otimes \cdots \otimes \mu_n$ to any n-dimensional distribution ν, assuming that (1) the transportation cost satisfies $\mathsf{c}(x, y) = \sum_i \mathsf{c}_i(x_i, y_i)$, where $x = (x_1, \ldots, x_n), y = (y_1, \ldots, y_n)$, and (2) the transporting algorithm A has oracle access to proper samplers for both μ, ν.

The algorithm is actually very simple: Given x, having determined $y_1, \ldots, y_{i-1}$, to determine y_i, it samples $k-1$ samples besides x_i according to μ_i. Similarly it samples k samples according to the conditional distribution of the ith coordinate of ν conditioned on the values of $y_1, \ldots, y_{i-1}$. Then it optimally matches the two sets of two k samples. The value of y_i is the match of x_i in this matching.

The transportation cost of A turns out to be $\Delta + \delta$, where Δ is the optimal cost of online transports from μ to ν (which, as we will prove, will coincide with the KR transport [27, 39] in our settings of interest), and δ is a term that could be made smaller by choosing k larger. We show that the reverse transport from ν back to the product μ can be done algorithmically as well. This will be useful for deriving further algorithmic transports through composition.

Sequential Samplers. When it comes to the samplability conditions needed in our main results above, we merely require that we can sample from μ_i efficiently. However, for the non-product distribution ν, the samplability condition is stronger and we require that one can sample from ν_i conditioned on any previously sampled prefix $y_{[i-1]}$. We refer to such samplers as *sequential* samplers. A key quantity of interest is the complexity of iterative sampling of the coordinates $y_1, \ldots, y_n$ sequentially (conditioned on previous ones) till we obtain a full sample y. We would like to have samplers where the average complexity of this sequential generation is bounded. As it turns out, we can indeed bound such costs in our special cases of interest.

From a real-world application point of view, this notion of efficient sequential sampler is very natural in some generative models. This is indeed the case for transformer-based language models that autoregressively generate their tokens one by one, each conditioned on the previously sampled sequence of tokens [44, 18]. That is, the joint distributions produced by these generative models have sequential samplers of low expected cost, as they indeed generate their sequence of symbols in a reasonable time and in an online fashion.

Algorithmic Transport for the Standard Gaussian Distribution. One of the fundamental results in OT is Talagrand's transportation inequality for the n-dimensional Gaussian distribution Φ^n [43]. It is proved that for every distribution ν, $\mathsf{T}(\Phi^n, \nu) \leq 2\mathsf{KL}(\nu, \Phi^n)$, in which the cost is measured in ℓ_2^2, i.e., $\mathsf{c}(x, y) = \sum_{i \in [n]} |x_i - y_i|^2$, and $\mathsf{KL}(\cdot, \cdot)$ denotes the Kullback–Leibler divergence. In this work, we lift this classical result to the algorithmic setting. Note that, as mentioned in [43], this bound is optimal *in general*, e.g., when ν is a shifted Φ^n, in which case our results converge to this optimal bound as well. In particular, we derive this result from our main result by proving the following two complementary claims:

- *Information theoretic:* We observe that Talagrand's bound of $2\mathsf{KL}(\nu, \Phi^n)$ upper bounds not only the best "offline" transport from the standard Gaussian Φ^n, but also the best *optimal online* transportation of Φ^n to ν. Namely, we show that $\Delta \leq 2\mathsf{KL}(\nu, \Phi^n)$, where Δ is the optimal online transportation cost as defined above.
- *Computational:* We use results from [19] to show that the Gaussian distribution in one dimension has a small transportation cost to its empirical samples on average.

Transporting Standard Gaussian to Conditional Gaussian. We show that in a natural setting, where ν is the Gaussian distribution conditioned on an event $\mathcal{S}$ of Gaussian measure ε, such sequential samplers can be efficiently simulated using oracle access to membership tests in $\mathcal{S}$. In other words, we find an algorithmic oracle-aided transportation algorithm that *simultaneously* work for all such distributions $\nu = \Phi^n|\mathcal{S}$. Note that such distributions have $2\mathsf{KL}(\nu, \Phi^n) \leq 2\ln 1/\varepsilon$. We obtain algorithmic transports running in expected time $\mathrm{poly}(n/\varepsilon)$ that achieve transport cost that converges to the upper bound of Talagrand.

Dimension-Independent Computational Concentration for Gaussian Spaces. One of the applications of OT is to obtain concentration of measure (CoM) inequalities [22]: One shows that any set $\mathcal{S}$ of "sufficiently large" measure in a concentrated metric probability space (μ, d), where μ is a distribution and d is a distance metric, expands to cover most of the measure in μ, when we consider neighbors of $\mathcal{S}$ within a certain distance. Recently, a *computational* (algorithmic) variant of the CoM phenomenon has been introduced [31, 15], in which one aims to show that the reverse mapping can be computed efficiently from almost all of the points in μ back to $\mathcal{S}$ by moving the points within a bounded distance. Namely, given a typical sampled point $x \sim \mu$, we aim to algorithmically find a "close neighbor" $y \in \mathcal{S}$ of bounded distance $\mathrm{d}(x, y)$. The work of [15] obtained such results for various settings, but their work left open obtaining computational CoM with dimension-independent (optimal) distance for the basic and natural space of Gaussian distributions under the ℓ_2 distance. Using our oracle set-transportation result for Gaussian spaces mentioned above, we resolve this open question and obtain such an optimal and dimension-free bound (see Corollary 36).

Reductions for (Deriving New) Algorithmic Transport. Finally, considering the role of reductions in resolving algorithmic tasks, we also develop the (right) notion of algorithmic reductions for the goal of relating algorithms for (optimal) transport across different spaces. In particular, suppose μ_1, μ_2 are distributions and $\mathsf{c}_1, \mathsf{c}_2$ are two different transportation costs. In the full version we state conditions under which, we can automatically transform an algorithmic transport result from μ_1 to ν (under the cost c_1) to a similar result that transports μ_2 to ν (under the cost c_2) for specific distributions μ_1, μ_2 and arbitrary distribution ν. We then show how to realize such reductions when we transport uniform distributions over the unit cube and the unit sphere (to an arbitrary distribution) by reducing them to the case of transporting Gaussian distributions. Consequently, we obtain algorithmic transports from these distributions as well.

2 Basic Concepts

In this section, we define the key notions studied in this paper and prove their basic properties.

Notation. We let $[n] = \{1, \ldots, n\}$. We denote the source (initial) distribution as μ. When μ is distributed over $\mathbb{R}^n$, we say that μ has dimension n and by μ_i we denote the distribution of its ith coordinate. We usually denote $x \sim \mu$, where $x = (x_1, \ldots, x_n)$ and $x_i \sim \mu_i$. $\mu = \mu_1 \otimes \cdots \otimes \mu_n$ means that μ is a product distribution. We use a similar notation for the target distribution ν. By $y \leftarrow A(x)$ we denote the process of running a probabilistic algorithm A on input x to obtain output y. When μ is a distribution, A^μ denotes an oracle algorithm A that has access to fresh samples from μ, and when $\mathcal{S}$ is a set, $A^{\mathcal{S}}$ denotes a similar situation where the oracle responds membership in $\mathcal{S}$. For vector $(v_1, \ldots, v_n)$, by $v_{[i]}$ we denote the prefix vector $(v_1, \ldots, v_i)$. When a distribution μ of dimension n with marginals $\mu_1, \ldots, \mu_n$ is clear from the context, by $\mu_i|x_{[i-1]}$, we denote the distribution of μ_i conditioned on having sampled x_j from μ_j for all $j < i$. For further clarity on the

underlying joint distribution, we might sometimes use $\mu_i|_\mu x_{[i-1]}$ instead. By $\mu(\mathcal{S})$ or $\Pr_\mu[\mathcal{S}]$ we denote the probability of event $\mathcal{S}$ under the distribution μ. Whenever it is clear from the context, for an outcome x, we use $\mu(x)$ to either denote the probability of the outcome x or the density of μ at x depending on whether μ is discrete or continuous. By $\mathrm{Supp}(\mu)$ we denote the support set of μ, which for the discrete and continuous cases can be defined as $\{x \mid \mu(x) > 0\}$. When $\mathrm{Supp}(\mu) \cup \mathrm{Supp}(\nu) \subseteq \mathcal{S}$, their Kullback–Leibler (KL) divergence is denoted as $\mathsf{KL}(\nu, \mu) = \sum_{x \in \mathcal{S}} \nu(x) \log(\nu(x)/\mu(x))$ with the natural logarithm basis. In the preceding definition and generally throughout this paper, we use the summation notation corresponding to discrete distributions; the corresponding results for continuous distributions replace sums with proper integrals. For $p \geq 1$, the ℓ_p-norm and ℓ_p-distance over $\mathbb{R}^n$ are defined as $\ell_p(x) = \|x\|_p = \left(\sum_{i \in [n]} |x_i|^p\right)^{1/p}$, and $\ell_p(x, y) = \ell_p(x - y)$.

Transportation Costs. In the following, all transportation *costs*, usually denoted as c, are functions $\mathsf{c} \colon \mathbb{R}^{2n} \mapsto \mathbb{R}$ with non-negative outputs that model the cost of transporting $x \sim \mu$ to $y \sim \nu$. We always assume c to be lower semi-continuous but do not assume c to be symmetric or satisfy the triangle inequality; we state these conditions, whenever needed.

▶ **Definition 1** (Coupling and Optimal Transportation Cost). *We say that a distribution π over pairs with marginals π_1, π_2 is a coupling of μ, ν if $\pi_1 = \mu, \pi_2 = \nu$. If for every $x \sim \mu$, there is a unique y such that $(x, y) \in \mathrm{Supp}(\pi)$, then we call this a deterministic (Monge) transport from μ to ν. For a cost c, the transport cost of a coupling π of μ, ν is defined as*

$$\mathsf{T}_\mathsf{c}(\pi) = \mathop{\mathbb{E}}_{(x,y)\sim\pi} \mathsf{c}(x, y).$$

We refer to $\mathsf{T}^{1/p}_{\mathsf{c}^p}(\pi)$ as the (Wasserstein) p-cost of π under c. If $\mathcal{C}(\mu, \nu)$ denotes the set of all couplings between μ, ν, the (Kantorovich) optimal transportation cost *for (μ, ν) is defined as*

$$\mathsf{T}_\mathsf{c}(\mu, \nu) = \inf_{\pi \in \mathcal{C}(\mu,\nu)} \mathsf{T}_\mathsf{c}(\pi).$$

The infimum in Definition 1 for defining the optimal transportation costs turns out to be a minimum as c is lower-semi continuous [1].

▶ **Definition 2** (Algorithmic Transport). *For distributions μ, ν, algorithm A is a* transport *from distribution μ to distribution ν if A is a (probabilistic) algorithm such that $A(x) \sim \nu$ whenever $x \sim \mu$. By π_A we denote the coupling created by A. For a transportation cost c the transportation cost of A is defined as $\mathsf{T}_\mathsf{c}(A) = \mathsf{T}_\mathsf{c}(\pi_A)$.*

Computational Model. In Definition 2, we need to either work with discrete distributions with samples of finite length, or when the distributions are continuous we need to work with the generalization of *algorithms working with real numbers* as formalized in [8, 9].

We now define an algorithmic variant of so-called *causal* transport [30] with a discrete time [3], We call it "online" to emphasize on the algorithmic aspect a la *online learning* [41].

▶ **Definition 3** (Online (Algorithmic) Transport). *For distributions μ, ν of dimension n, we call a (probabilistic and perhaps computationally unbounded) algorithm A an* online transport *algorithm from μ to ν if it forms a transport from μ to ν, while it makes its decisions in an online way. Namely, A has an internal iterating process (for simplicity also denoted by A) that reads $(x_1, \ldots, x_n) \sim \mu$ coordinate by coordinate while holding an internal state, initially $s_0 = \varnothing$. In the ith iteration, we have $(s_i, y_i) \leftarrow A(s_{i-1}, x_i)$, and at the end we output $(y_1, \ldots, y_n) \sim \nu$. We also let $\mathcal{C}^{\mathrm{OnT}}(\mu, \nu)$ to be the set of all couplings that can be obtained by online algorithms and for a transport cost c obtain the optimal online transportation cost as*

$$\mathsf{T}^{\mathrm{OnT}}_\mathsf{c}(\mu, \nu) = \inf_{\pi \in \mathcal{C}^{\mathrm{OnT}}(\mu,\nu)} \mathsf{T}_\mathsf{c}(\pi).$$

To contrast and emphasize on a transport not being necessarily online, we refer to (potentially) non-online transports as *offline* transports.

We now define a class of couplings that is closely related to online transport.

▶ **Definition 4** (Online Coupling). *Suppose π is a coupling between n-dimensional distributions μ, ν, and π_i is the corresponding marginal coupling between μ_i, ν_i. We call π an* online coupling *if for all $z = (x_{[i-1]}, y_{[i-1]}) \in \mathrm{Supp}(\pi_{[i-1]})$, $\pi_i|z$ is a coupling of $\mu_i|x_{[i-1]}$ (according to μ) and $\nu_i|y_{[i-1]}$ (according to ν). If $\mathcal{C}^{\mathrm{OnC}}(\mu,\nu)$ denotes the set of all online couplings between μ, ν, for a transport cost c we obtain the optimal online coupling cost between μ, ν as*

$$\mathsf{T}_{\mathsf{c}}^{\mathrm{OnC}}(\mu,\nu) = \inf_{\pi \in \mathcal{C}^{\mathrm{OnC}}(\mu,\nu)} \mathsf{T}_{\mathsf{c}}(\pi).$$

We now show how to characterize online couplings using online transports.

▶ **Proposition 5.** *A coupling π between μ, ν is* online *if and only if it can be obtained through both an online transport from μ to ν and an online transport from ν to μ.*

▶ **Definition 6.** *We call the cost function c over $\mathbb{R}^n \times \mathbb{R}^n$* linear *over $\mathsf{c}_1, \ldots, \mathsf{c}_n$, if $\mathsf{c}(x,y) = \mathsf{c}_1(x_1,y_1) + \cdots + \mathsf{c}_n(x_n,y_n)$, for all $x = (x_1,\ldots,x_n), y = (y_1,\ldots,y_n)$.*

Greedy Coupling. One might wonder how we can compute/approximate $\mathsf{T}_{\mathsf{c}}^{\mathrm{OnC}}(\mu,\nu)$. One approach is to use greedy methods, by trying to use an optimal coupling in each round. This is formalized in the following definition in settings with dedicated costs for each round. We will then discuss when this method succeeds in Theorem 10. More generally, we define locally-optimal couplings, even when they are not online.

▶ **Definition 7** (Locally Optimal and Greedy Couplings). *Suppose the cost function c over $\mathbb{R}^{2n}$ is linear over $\mathsf{c}_1, \ldots, \mathsf{c}_n$. A coupling π between μ, ν is* locally optimal, *if for every $z_{[i-1]} \in \mathrm{Supp}(\pi_{[i-1]})$, it holds that $\pi_i|z_{[i-1]}$ is an OT between $\mu_i|z_{[i-1]}, \nu_i|z_{[i-1]}$; i.e.,*

$$\mathsf{T}_{\mathsf{c}_i}(\pi_i|z_{[i-1]}) = \mathsf{T}_{\mathsf{c}_i}(\mu_i|z_{[i-1]}, \nu_i|z_{[i-1]}).$$

When π is an online coupling as well, the above condition simplifies to $\mathsf{T}_{\mathsf{c}_i}(\pi_i|z_{[i-1]}) = \mathsf{T}_{\mathsf{c}_i}(\mu_i|x_{[i-1]}, \nu_i|y_{[i-1]})$ in which case we call π greedy. *For $\mathcal{C}^{\mathrm{G}}(\mu,\nu)$ denoting the set of all greedy couplings from μ to ν, we define*

$$\mathsf{T}_{\mathsf{c}}^{\mathrm{G}}(\mu,\nu) = \sup_{\pi \in \mathcal{C}^{\mathrm{G}}(\mu,\nu)} \mathsf{T}_{\mathsf{c}}(\pi).$$

▶ Remark 8 (Greedy vs. Knothe-Rosenblatt Transports). Greedy couplings are closely related to Knothe-Rosenblatt (KR for short) transports [27, 39]. Specifically, for a greedy coupling π, when the cost functions c_i are convex, for any $z_{[i-1]} \sim \pi_{[i-1]}$, the locally optimal coupling $\pi_i|z_{[i-1]}$ could be obtained by simply using the unique monotone mapping [12]. Hence, KR coupling is a special case of greedy couplings and cover many interesting cases in this class. For example, when the cost function c is ℓ_p^p for $p \geq 1$, then $\mathsf{T}_{\mathsf{c}}^{\mathrm{G}}(\mu,\nu)$ equals the cost of the KR coupling between μ and ν. However, due to the generality of greedy couplings (e.g., for non-monotone costs) we define and use greedy transports.

Lambda and Delta Cost Functions. We now define two functions that play key roles in our analysis of the cost of online transports. The first (Lambda) function depends on a coupling, while the second one (Delta) depends on the two distributions that are coupled. As we prove later in Proposition 12, Lambda is a parameter that lower bounds the cost of any coupling. Delta is the optimal online transport from a product distribution to another one.

▶ **Definition 9** (The Lambda and Delta Functions). *For a coupling π of dimension n between distributions μ, ν of dimension n, and a cost function c that is linear over $\mathsf{c}_1, \ldots, \mathsf{c}_n$, we define the* Lambda functions *as*

$$\Lambda_{\mathsf{c}}(\pi) = \mathop{\mathbb{E}}_{z\sim\pi} \sum_{i\in[n]} \mathsf{T}_{\mathsf{c}_i}(\mu_i|z_{[i-1]}, \nu_i|z_{[i-1]}).$$

We also define the Delta function *between distributions μ, ν of dimension n as*

$$\Delta_{\mathsf{c}}(\mu, \nu) = \mathop{\mathbb{E}}_{y\sim\nu} \sum_{i\in[n]} \mathsf{T}_{\mathsf{c}_i}(\mu_i, \nu_i|y_{[i-1]}).$$

Note that the coupling π in Definition 9 does not have to be online. Furthermore, the definition of $\Lambda(\cdot)$ does depend on the order of the coordinates of the n-dimension distributions.

2.1 Online Coupling and Transport from Products

We end this section by stating a theorem showing that, whenever μ is product, any online coupling that is "locally optimal" in the sense that given history $z = (x_{[i-1]}, y_{[i-1]})$ it finds (an arbitrary) optimal transport between $(\mu_i), (\nu_i|y_{[i-1]})$, finds an optimum online coupling between μ, ν as well as an optimal online transport from μ to ν. This theorem does not assume convexity of the costs. As stated in Remark 8, for convex transportation costs, greedy algorithms can be instantiated using the KR transform.

▶ **Theorem 10** (Optimal Online Coupling and Transport from Products). *If $\mu = \mu_1 \otimes \cdots \otimes \mu_n$ is product and the cost function c is linear over $\mathsf{c}_1, \ldots, \mathsf{c}_n$, then*

$$\mathsf{T}_{\mathsf{c}}^{\mathrm{OnT}}(\mu, \nu) = \mathsf{T}_{\mathsf{c}}^{\mathrm{OnC}}(\mu, \nu) = \mathsf{T}_{\mathsf{c}}^{\mathrm{G}}(\mu, \nu) = \Delta_{\mathsf{c}}(\mu, \nu).$$

Before proving Theorem 10 we prove some basic tools that are used in the proof. The first lemma that we state can be obtained from a simple application of the linearity of expectation.

▶ **Lemma 11** (Cost Splitting). *Let π be a coupling between distributions μ, ν of dimensions n, and let π_i be the corresponding coupling between the marginals μ_i, ν_i. Suppose c is linear over $\mathsf{c}_1, \ldots, \mathsf{c}_n$, and ω is an n-dimensional distribution that is arbitrarily correlated with π. Then,*

$$\mathsf{T}_{\mathsf{c}}(\pi) = \sum_{i\in[n]} \mathsf{T}_{\mathsf{c}_i}(\pi_i) = \mathop{\mathbb{E}}_{z\sim\omega} \sum_{i\in[n]} \mathsf{T}_{\mathsf{c}_i}(\pi_i|\omega_{[i-1]} = z_{[i-1]}).$$

In particular, we can choose $\omega = \nu$, $\omega = \mu$, or $\omega = \pi$ as special cases.

We now prove some basic properties of the two functions, showing how to use it and how to characterize it in some special settings. In summary, Lambda function lower bounds the transportation of every coupling, while Delta will play a key role in characterizing the transportation cost for product distributions.

▶ **Proposition 12** (Properties of Lambda and Delta Functions). *Suppose π couples μ, ν and c is linear. The Lambda function satisfies the following properties.*

1. Lower Bound: *For all π, $\Lambda_{\mathsf{c}}(\pi) \le \mathsf{T}_{\mathsf{c}}(\pi)$, and the equality holds iff π is locally optimal.*
2. Online Transports from Products: *If π is an online transport and $\mu = \mu_1 \otimes \cdots \otimes \mu_n$, then*

   $$\Lambda_{\mathsf{c}}(\pi) \ge \Delta_{\mathsf{c}}(\mu, \nu).$$

3. Online Coupling for Products: *If π is an online* coupling, *and μ is product then*

   $$\Lambda_{\mathsf{c}}(\pi) = \Delta_{\mathsf{c}}(\mu, \nu).$$

Proof of Proposition 12. We prove the claims in order.

1. By letting $\omega = \pi$ in Lemma 11, we get

$$\mathsf{T}_{\mathsf{c}}(\pi) = \mathop{\mathbb{E}}_{z\sim\pi} \sum_{i\in[n]} \mathsf{T}_{\mathsf{c}_i}(\pi_i|z_{[i-1]}) \geq \mathop{\mathbb{E}}_{z\sim\pi} \sum_{i\in[n]} \mathsf{T}_{\mathsf{c}_i}(\mu_i|z_{[i-1]}, \nu_i|z_{[i-1]}) = \Lambda_{\mathsf{c}}(\pi),$$

   where the inequality follows from the fact that $\mathsf{T}_{\mathsf{c}_i}(\cdot,\cdot)$ minimizes the transportation cost.
2. We first claim that, in this case, for every $z_{[i-1]} = (x_{[i-1]}, y_{[i-1]}) \sim \pi_{[i-1]}$, we have $\mu_i|z_{[i-1]} = \mu_i$. This is true, because (1) $(\mu_i|x_{[i-1]}, y_{[i-1]}) = (\mu_i|x_{[i-1]})$ and the fact that π is an online transport, and (2) $(\mu_i|x_{[i-1]}) = \mu_i$ by the fact that μ is a product. Therefore,

$$\Lambda_{\mathsf{c}}(\pi) = \mathop{\mathbb{E}}_{z\sim\pi} \sum_{i\in[n]} \mathsf{T}_{\mathsf{c}_i}(\mu_i|z_{[i-1]}, \nu_i|z_{[i-1]}) = \mathop{\mathbb{E}}_{z=(x,y)\sim\pi} \sum_{i\in[n]} \mathsf{T}_{\mathsf{c}_i}(\mu_i, \nu_i|z_{[i-1]}).$$

   We now use the right hand side. In analyzing the right hand side, we first use Lemma 11 (using $\omega = \pi$) and then sample x, y in reverse order,

$$\mathop{\mathbb{E}}_{(x,y)\sim\pi} \sum_{i\in[n]} \mathsf{T}_{\mathsf{c}_i}(\mu_i, \nu_i|z_{[i-1]}) = \sum_{i\in[n]} \mathop{\mathbb{E}}_{y_{[i-1]}\sim\nu_{[i-1]}} \mathop{\mathbb{E}}_{x_{[i-1]}\sim\nu_{[i-1]}|y_{[i-1]}} \mathsf{T}_{\mathsf{c}_i}(\mu_i, \nu_i|y_{[i-1]}, x_{[i-1]}),$$

   where for each $i \in [n]$, we sample $(x_{[i-1]}, y_{[i-1]}) \sim \pi_{[i-1]}$ by first sampling $y_{[i-1]}$ and then sampling $x_{[i-1]}$ conditioned on $y_{[i-1]}$. Now, for every $y_{[i-1]} \sim \nu_{[i-1]}$, we claim that

$$\mathop{\mathbb{E}}_{x_{[i-1]}\sim\nu_{[i-1]}|y_{[i-1]}} \mathsf{T}_{\mathsf{c}_i}(\mu_i, \nu_i|y_{[i-1]}, x_{[i-1]}) \geq \mathsf{T}_{\mathsf{c}_i}(\mu_i, \nu_i|y_{[i-1]}).$$

   This claim follows from Part 2 of Proposition 19 and the fact that the average of $\nu_i|y_{[i-1]}, x_{[i-1]}$ over the choice of $x_{[i-1]} \sim \nu_{[i-1]}|y_{[i-1]}$ is $\nu_i|y_{[i-1]}$.
3. When the coupling π is further an online coupling, then the equality holds, because $(\nu_i|y_{[i-1]}, x_{[i-1]}) = (\nu_i|y_{[i-1]})$, and the last inequality above becomes an equality. ◀

Proof of Theorem 10. It is enough to prove the following two claims.

1. $\mathsf{T}^{\mathrm{G}}_{\mathsf{c}}(\mu,\nu) \leq \Delta_{\mathsf{c}}(\mu,\nu)$.
2. $\mathsf{T}^{\mathrm{OnT}}_{\mathsf{c}}(\mu,\nu) \geq \Delta_{\mathsf{c}}(\mu,\nu)$.

The reason is that we already know $\mathsf{T}^{\mathrm{OnT}}_{\mathsf{c}}(\mu,\nu) \leq \mathsf{T}^{\mathrm{G}}_{\mathsf{c}}(\mu,\nu)$ (as being greedy is a limitation), and so proving the two claims above would imply all the equalities of the theorem statement.

To prove the first claim, we observe that cost $\Delta_{\mathsf{c}}(\mu,\nu)$ can be achieved using (any) greedy algorithm that (by definition) optimally couples $\mu_i = \mu_i|x_{[i-1]}$ with $\nu_i|y_{[i-1]}$ in the ith step. In fact, *all* greedy coupling algorithms have the same cost $\Delta_{\mathsf{c}}(\mu,\nu)$ when one of the distributions is product.

To prove the second claim, let π be an online transport with cost $\mathsf{T}^{\mathrm{OnT}}_{\mathsf{c}}(\mu,\nu)$. Our claim follows from Parts 1 and 2 of Proposition 12, due to π being online and μ being a product.

$$\mathsf{T}^{\mathrm{OnT}}_{\mathsf{c}}(\mu,\nu) = \mathsf{T}_{\mathsf{c}}(\pi) \geq \Lambda_{\mathsf{c}}(\pi) \geq \Delta_{\mathsf{c}}(\mu,\nu).$$ ◀

3 Basic Tools

3.1 Composition and Triangle Inequalities

Multi-distribution Coupling and Composition. We now generalize the notion of coupling to more than two distributions and use it to define composition of (online) couplings.

▶ **Definition 13** (Multi-distribution Coupling). *A coupling π of $\mu_1, \ldots, \mu_n$ is a distribution over n-vectors such that the marginal of the ith coordinate is distributed as μ_i.*

▶ **Definition 14** (Composition of Couplings). *For coupling $\pi_{1,2}$ over μ_1, μ_2 and coupling $\pi_{2,3}$ over μ_2, μ_3, we define the composition $\pi_{1,3} = \pi_{2,3} \circ \pi_{1,2}$ of $\pi_{1,2}$ and $\pi_{2,3}$ as the marginal of the first and third coordinates of the (unique) coupling of μ_1, μ_2, μ_3 such that.*

1. *For $1 \le i < j \le 3$, the marginal distribution of (μ_i, μ_j) in $\pi_{1,2,3}$ is distributed as $\pi_{i,j}$.*
2. *In the coupling $\pi_{1,2,3}$, μ_1, μ_3 are independent, conditioned on $x_2 \sim \mu_2$.*

We now use Wasserstein p-cost, to state the following well-known triangle inequality.

▶ **Lemma 15** (Triangle Inequality for Wasserstein p-Costs). *Suppose a cost function c satisfies the triangle inequality (but not necessarily symmetry) and $p \ge 1$. Then, for every coupling π over μ_1, μ_2, μ_3 with marginal coupling $\pi_{i,j}, i < j$ over π_i, π_j, we have the following,*

$$\mathsf{T}^{1/p}_{\mathsf{c}^p}(\pi_{1,3}) \le \mathsf{T}^{1/p}_{\mathsf{c}^p}(\pi_{1,2}) + \mathsf{T}^{1/p}_{\mathsf{c}^p}(\pi_{2,3}).$$

The following proposition can be obtained from the triangle inequality of Lemma 15.

▶ **Proposition 16** (Triangle Inequality for Wasserstein p-Costs in Multi-Round Settings). *Let μ be a distribution over $\mathbb{R}^n$, and for every $i \in [n], x_{[i-1]} \in \mathrm{Supp}(\mu_{[i-1]})$ let $J(x_{[i-1]})$ be a distribution over* triples *of distributions over $\mathbb{R}$. Suppose c satisfies the triangle inequality and c^p is linear over $\mathsf{c}_1, \ldots, \mathsf{c}_n$ for $p \ge 1$. Then, the following holds.*

$$\left(\mathop{\mathbb{E}}_{x\sim\mu} \sum_{i\in[n]} \mathop{\mathbb{E}}_{(\nu_1,\nu_2,\nu_3)\sim J(x_{[i-1]})} \mathsf{T}_{\mathsf{c}_i}(\nu_1, \nu_3) \right)^{1/p}$$
$$\le \sum_{k\in[2]} \left(\mathop{\mathbb{E}}_{x\sim\mu} \sum_{i\in[n]} \mathop{\mathbb{E}}_{(\nu_1,\nu_2,\nu_3)\sim J(x_{[i-1]})} \mathsf{T}_{\mathsf{c}_i}(\nu_k, \nu_{k+1}) \right)^{1/p}$$

The following can be obtained from the definition of online transport and Lemma 15.

▶ **Lemma 17** (Properties of the Composition of Online Transports). *Consider an online transport $A_{1,2}$ from μ_1 to μ_2 with coupling $\pi_{1,2}$ and an online transport $A_{2,3}$ from μ_2 to μ_3 with coupling $\pi_{2,3}$. Let $\pi_{1,3} = \pi_{2,3} \circ \pi_{1,2}$ be the composed coupling. Then,*

1. *The coupling $\pi_{1,3}$ is an online coupling.*
2. *There is an algorithm $A_{1,3}$ that transports μ_1 to μ_3 as the coupling $\pi_{1,3}$, whose complexity is bounded by running $A_{1,2}$ followed by running $A_{2,3}$.*
3. *If the cost function c satisfies the triangle inequality, then for all $p \ge 1$ the following holds*

$$\mathsf{T}^{1/p}_{\mathsf{c}^p}(A_{1,3}) \le \mathsf{T}^{1/p}_{\mathsf{c}^p}(A_{1,2}) + \mathsf{T}^{1/p}_{\mathsf{c}^p}(A_{2,3}).$$

The first item in Lemma 17 and Proposition 5 together show that the composition of two online coupling is also an online coupling.

3.2 Transport Through Intermediate Distributions

In this section, we describe a method of transporting μ to ν (perhaps in an online and iterative way) through optimal transports between intermediate distributions in one dimension. We start with some definitions. We start by defining the notion of average for distributions and stating a general way of transporting through averages.

▶ **Definition 18** (Average Distribution). *Suppose M is a distribution over distributions. We define* the average *of M, denoted as $\mathbb{E}[M] = \mathbb{E}_{\mu'\sim M}[\mu'] = \mu$, to be the distribution μ of the random variable x that is sampled by first sampling $\mu' \sim M$ and then $x \sim \mu'$. Namely, μ is the distribution that $\mu(\mathcal{S}) = \mathbb{E}_{\mu'\sim M}\, \mu'(\mathcal{S})$ for all the events $\mathcal{S}$ defined over $\cup_{\mu'\in\mathrm{Supp}(M)} \mathrm{Supp}(\mu')$.*

▶ **Proposition 19** (Transport to Averages). *Suppose M is a distribution over distributions with average μ.*

1. *Suppose π is the following joint distribution. We first sample $\mu' \sim M$, then couple μ' with ν as $\pi_{\mu'}$, and then output a sample $(x, y) \sim \pi_{\mu'}$. Then, π is a coupling between μ, ν.*
2. $\mathbb{E}_{\mu' \sim M} \mathsf{T}_{\mathsf{c}}(\mu', \nu) \geq \mathsf{T}_{\mathsf{c}}(\mu', \nu)$.

Proof. Part 1 holds because the marginals of x and y have the marginals of μ, ν. Part 2 follows from Part 1 and picking $\pi_{\mu'}$ to be the optimal transport between μ', ν. ◀

The following definition states a way of finding a transport from μ to ν by working with alternative (intermediate) distributions that approximate μ, ν.

▶ **Definition 20** (Transport Through Intermediate Distributions). *Let μ, ν be distributions, c be a cost function, and J be a distribution over pairs of distributions. We say that algorithm A couples μ, ν* through (the intermediate distribution) *J, if the following conditions hold.*

1. *J produces marginals with averages μ, ν. I.e., $\mu = \mathbb{E}_{(\mu', \nu') \sim J} \mu'$ and $\nu = \mathbb{E}_{(\mu', \nu') \sim J} \nu'$.*
2. *Algorithm A first samples $(\mu', \nu') \sim J$, then finds some* optimal *transport π between μ', ν' according to c, and finally outputs $(x, y) \sim \pi$.*

▶ **Definition 21** (Conditioning and Composing Transports with Distributions). *Suppose μ', μ, ν are distributions and π is a transport from μ to ν. If $\mathrm{Supp}(\mu') \subseteq \mathrm{Supp}(\mu)$, then consider the following sampling process.*

1. *Sample $x \sim \mu'$.*
2. *Sample y from the ν-coordinate of π, conditioned on its μ-coordinate being x.*

Then, the notation $\pi|\mu'$ denotes the joint distribution of (x, y) and $\pi \sharp \mu'$ denotes the distribution of y. Additionally, if M is a distribution over distributions, then $N = \pi \sharp M$ denotes the distribution over distributions sampled by outputting $\nu' = \pi \sharp \mu'$ for $\mu' \sim M$.

Notation. Let $U_{k,\mu}$ be the distribution over distributions obtained by first sampling $\mathcal{X} \sim \mu^k$, and then outputting $\mu' = U_{\mathcal{X}}$. A simple observation is that $\mathbb{E}\, U_{k,\mu} = \mu$ for all k.

▶ **Proposition 22.** *If M is a distribution over distributions with average distribution μ, and if π is any transport from μ to ν, then the following holds.*

1. *$N = \pi \sharp M$ is a distribution over distributions with average ν.*
2. *For cost c, $\mathsf{T}_{\mathsf{c}}(\pi) = \mathbb{E}_{\mu' \sim M} \mathsf{T}_{\mathsf{c}}(\pi|\mu')$ in which $\pi|\mu'$ is defined in Definition 21.*
3. *$U_{k,\nu} = \pi \sharp U_{k,\mu}$, and if μ is samplable in time t_μ and coupling π is computable in time t_π, then one can sample the set $\mathcal{Y}, |\mathcal{Y}| = k$ that describes $U_{\mathcal{Y}} \sim U_{k,\nu}$ in time $k(t_\mu + t_\pi)$.*

Proof. For Part 1, observe that if we sample $x \sim \mu'$ for $\mu' \sim M$, by definition we get $x \sim \mu$, which means $y \sim \pi \sharp M$ will be sampled as $y \sim \nu$. For Part 2, $\mathbb{E}_{\mu' \sim M} \mathsf{T}_{\mathsf{c}}(\pi|\mu')$ also computes the cost of the same coupling π by breaking it into marginal costs based on how $x \sim \mu$ is sampled. For Part 3, let $(x, y) \sim \pi$. We first sample $(x_1, \ldots, x_k) = \mathcal{X} \sim \mu^k$ and then let $\mathcal{Y} = (y_1, \ldots, y_k)$ for $y_i \sim y|x = x_i$. It holds that x_is are independently sampled according to μ, and because π transports μ to ν, y_i's are also independently sampled according to ν. ◀

▶ **Lemma 23** (Multi-Round Algorithmic Coupling Through Intermediate Distributions). *Suppose cost function c satisfies the triangle inequality, and c^p is linear over $\mathsf{c}_1, \ldots, \mathsf{c}_n$ for $p \geq 1$. Let π, with marginals $\pi_1, \ldots, \pi_n$ be a transport from μ with marginals $\mu_1, \ldots, \mu_n$ to ν with marginals $\nu_1, \ldots, \nu_n$. For round $i \in [n]$ and previously sampled $z_{[i-1]} = (x_{[i-1]}, y_{[i-1]}) \in \mathrm{Supp}(\pi_{[i-1]})$, suppose $J(z_{[i-1]})$ is a distribution over pairs of distributions defined based on $z_{[i-1]}$, and $\sigma_{z_{[i-1]}}$ is an* optimal *transport from $\mu_i|z_{[i-1]}$ to $\nu_i|z_{[i-1]}$ under c_i. Suppose π can also be*

obtained using the following algorithm A in n rounds. In round $i \in [n]$ and for previously sampled $z_{[i-1]} = (x_{[i-1]}, y_{[i-1]}) \in \mathrm{Supp}(\pi_{[i-1]})$, A couples $\mu_i|z_{[i-1]}$ and $\nu_i|z_{[i-1]}$ through the intermediate distribution $J(z_{[i-1]})$ as defined in Definition 20 using the cost c_i. Then,

$$\mathsf{T}^{1/p}_{\mathsf{c}^p}(\pi) \leq \left(\mathop{\mathbb{E}}_{z\sim\pi} \sum_{i\in[n]} \mathop{\mathbb{E}}_{(\mu'_i,\nu'_i)\sim J(z_{[i-1]})} \mathsf{T}_{\mathsf{c}_i}\left(\mu'_i, \sigma^{-1}_{z_{[i-1]}} \sharp \nu'_i\right) \right)^{1/p} + \Lambda^{1/p}_{\mathsf{c}^p}(\pi),$$

where σ^{-1} refers to the inverse coupling that changes the order of its marginals.

Proof of Lemma 23. The proof uses the triangle inequality for Wasserstein p-costs for the multi-round setting (Proposition 16).

For each $i \in [n]$ and $z_{[i-1]} \in \mathrm{Supp}(\pi_{[i-1]})$, consider the following sampling process $I(z_{[i-1]})$ that extends $J(z_{[i-1]})$ by outputting one more coordinate as well.

1. Sample $(\mu', \nu') \sim J(z_{[i-1]})$.
2. Let $\mu'' = \sigma^{-1}_{z_{[i-1]}} \sharp \nu'$.
3. Obtain $(\mu', \mu'', \nu') \sim I(z_{[i-1]})$.

It holds that $\mathsf{T}^{1/p}_{\mathsf{c}^p}(A) = \left(\mathbb{E}_{z\sim\pi} \sum_{i\in[n]} \mathbb{E}_{(\mu',\mu'',\nu')\sim I(z_{[i-1]})} \mathsf{T}_{\mathsf{c}_i}(\mu', \nu')\right)^{1/p}$, which is the left side of the inequality of Proposition 16, and the right side is:

$$\left(\mathop{\mathbb{E}}_{z\sim\pi} \sum_{i\in[n]} \mathop{\mathbb{E}}_{(\mu',\mu'',\nu')\sim I(z_{[i-1]})} \mathsf{T}_{\mathsf{c}_i}(\mu', \mu'') \right)^{1/p} + \left(\mathop{\mathbb{E}}_{z\sim\pi} \sum_{i\in[n]} \mathop{\mathbb{E}}_{(\mu',\mu'',\nu')\sim I(z_{[i-1]})} \mathsf{T}_{\mathsf{c}_i}(\mu'', \nu') \right)^{1/p}$$

The first term is exactly the first term on the right hand side of the inequality of the lemma. Therefore, all we have to do is to prove that

$$\mathop{\mathbb{E}}_{z\sim\pi} \sum_{i\in[n]} \mathop{\mathbb{E}}_{(\mu',\mu'',\nu')\sim I(z_{[i-1]})} \mathsf{T}_{\mathsf{c}_i}(\mu'', \nu') \leq \mathop{\mathbb{E}}_{z\sim\pi} \sum_{i\in[n]} \mathsf{T}_{\mathsf{c}_i}(\sigma_{z_{[i-1]}}).$$

In fact, we prove this statement for *every* choice of z and i, so ignoring z, i we prove the claim:

$$\mathop{\mathbb{E}}_{(\mu'',\nu')\sim I} \mathsf{T}_{\mathsf{c}_i}(\mu'', \nu') \leq \mathop{\mathbb{E}}_{(\mu'',\nu')\sim I} \mathsf{T}_{\mathsf{c}_i}((\sigma^{-1}|\nu')^{-1}) = \mathsf{T}_{\mathsf{c}_i}(\sigma),$$

where the middle term is added for the proof.

We now prove both the inequality and the equality above through the steps below.

- *Equality:* Since the average of $\nu' \sim J$ is ν_i and σ^{-1} is a transport from ν_i to μ_i, if we define $\mathsf{c}'_i(y_i, x_i) = \mathsf{c}_i(x_i, y_i)$, then by Part 2 of Proposition 22 we have

$$\mathop{\mathbb{E}}_{(\mu'',\nu')\sim I} \mathsf{T}_{\mathsf{c}_i}((\sigma^{-1}|\nu')^{-1}) = \mathop{\mathbb{E}}_{(\mu'',\nu')\sim I} \mathsf{T}_{\mathsf{c}'_i}(\sigma^{-1}|\nu') = \mathsf{T}_{\mathsf{c}'_i}(\sigma^{-1}) = \mathsf{T}_{\mathsf{c}_i}(\sigma).$$

- *Inequality:* Again, using $\mathsf{c}'_i(y_i, x_i) = \mathsf{c}_i(x_i, y_i)$, we have

$$\begin{aligned} \mathop{\mathbb{E}}_{(\mu'',\nu')\sim I} \mathsf{T}_{\mathsf{c}_i}(\mu'', \nu') &= \mathop{\mathbb{E}}_{(\mu'',\nu')\sim I} \mathsf{T}_{\mathsf{c}'_i}(\nu', \mu'') \\ &\leq \mathop{\mathbb{E}}_{(\mu'',\nu')\sim I} \mathsf{T}_{\mathsf{c}'_i}(\sigma^{-1}|\nu') = \mathop{\mathbb{E}}_{(\mu'',\nu')\sim I} \mathsf{T}_{\mathsf{c}_i}((\sigma^{-1}|\nu')^{-1}), \end{aligned}$$

where the inequality is due to the fact that $\mathsf{T}_{\mathsf{c}'_i}(\nu', \mu'')$ is the optimal cost. ◀

3.3 Borrowed Tools

The following can be obtained from the proofs in [43, 22] (see the full version). For $p = 2$, it gives the celebrated Talagrand's transportation inequality for Gaussian under ℓ_2.

▶ **Theorem 24** (Talagrand's Inequality for the Gaussian Measure). *If* $\mathsf{c}(x, y) = \ell_p^p(x, y)$, $p \in [1, 2]$, Φ_n *is the standard Gaussian and* ν *is an arbitrary distribution both in* $\mathbb{R}^n$, *then*

$$\mathsf{T}_{\mathsf{c}}^{\mathrm{OnT}}(\Phi_n, \nu) = \Delta_{\mathsf{c}}(\Phi_n, \nu) \le n^{1-p/2} \cdot (2\mathsf{KL}(\nu, \Phi_n))^{p/2}.$$

▶ **Definition 25** (Transports to Empirical). *For distributions* μ *and symmetric cost* c, *we let* $\mathsf{T}_{\mathsf{c},k}^{\mathrm{Em}}(\mu) = \mathbb{E}_{\mathcal{X}\sim\mu^k} \mathsf{T}_{\mathsf{c}}(U_{\mathcal{X}}, \mu)$ *denote the cost of transporting* μ *to an empirical set of size* k, *where* $U_{\mathcal{X}}$ *is the uniform distribution over the multi-set* $\mathcal{X}$.

The following lemma follows from [19] and known moments of the Gaussian distribution.

▶ **Lemma 26** (Original-to-Empirical Transport for the Normal Distribution). *Let* $p \ge 1$, c *be* ℓ_p^p, *and* $\mu = \mathcal{N}(0, 1)$ *is the normal distribution. Then, for a constant* C_p *depending on* p,

$$\mathsf{T}_{\mathsf{c},k}^{\mathrm{Em}}(\mu) \le C_p \cdot 2^{1+3p/2} \cdot \Gamma(p+1)^{\frac{p}{2p+1}} \cdot k^{-1/2}.$$

4 Algorithmic Transport for Products

In this section, we put together the tools from previous sections to derive algorithmic results about online transport for the setting that one of the source or target distributions is product. We then derive a corollary for the Gaussian measure. We first define sequential samplers.

▶ **Definition 27** (Sequential Sampler). *For a distribution* ν *in dimension* n *with marginals* $\nu_1, \ldots, \nu_n$, *we call* $\hat{\nu}$ *its* sequential sampler *for* ν, *if for* all $y_{[i-1]} \sim \nu_{[i-1]}$ *calling* $\hat{\nu}(y_{[i-1]})$ *returns an independent answer* $\hat{\nu}(y_{[i-1]}) \sim \nu_i | y_{[i-1]}$. *For queries* $y_{[i-1]} \notin \mathrm{Supp}(\nu_{[i-1]})$, *calling* $\hat{\nu}(y_{[i-1]})$ *returns* $\perp$. *We also assign a (sequential sampling) cost* $\mathsf{sc}_\nu(y_{[i-1]})$ *to query* $y_{[i-1]}$, *and call* $\mathsf{sc}_\nu = \mathbb{E}_y \sum_{i\in[n-1]} \mathsf{sc}_\nu(y_{[i-1]})$ *the* average (sequential sampling) cost *of* $\hat{\nu}$. *For an oracle-algorithm* A *calling (a potentially randomized) set* $\mathcal{Q}$ *of queries to* $\hat{\nu}$, *we define its* average total cost *of calling* $\hat{\nu}$ *as* $\mathsf{sc}_\nu^A = \mathbb{E}_{\mathcal{Q}} \sum_{a\in\mathcal{Q}} \mathsf{sc}_\nu(a)$.[1]

One natural way of using sc is to model sampling time, but it can model other costs as well. The average cost sc_ν of $\hat{\nu}$ is indeed the average total cost of the following simple algorithm A that uses sc_ν sequentially to obtain a full sample: Let $y_{[0]}$ be the empty string, and for $i \in [n]$, A let $y_i = \hat{\nu}(y_{[i-1]})$. Also, when μ is a product distribution, then $\hat{\mu}$ is nothing other than a direct way of sampling from independent distributions ν_i for all $i \in [n]$.

Before stating our main result, recall the notation for transport cost to empirical sets from Definition 25.

▶ **Theorem 28** (Main Result). *Suppose* $\mu = \mu_1 \otimes \cdots \otimes \mu_n$ *and* ν *are distributions over* $\mathbb{R}^n$, *with sequential samplers* $\hat{\mu}, \hat{\nu}$ *and corresponding oracle cost functions* $\mathsf{sc}_\mu, \mathsf{sc}_\nu$. *Suppose the transportation cost function* c *is a metric (i.e., symmetric and satisfies the triangle inequality) and* c^p *is linear over symmetric costs* $\mathsf{c}_1, \ldots, \mathsf{c}_n$.[2] *Then, there is an algorithm* A_k, *parameterized by* k, *that uses oracle access to samplers* $\hat{\mu}, \hat{\nu}$ *and achieves the following:*

[1] Since $\mathsf{sc}_\nu(y_{[i-1]})$ naturally measures the (e.g., computational) cost of sampling a coordinate conditioned on previously sampled coordinates, for natural settings and independent ν_1, ν_2, the value of $\mathsf{sc}_\nu(y_{[1]})$ will be independent of $y_{[1]}$.

[2] An example is $\mathsf{c} = \ell_p$.

1. $A_k^{\hat{\mu},\hat{\nu}}$ *transports* μ *to* ν *through an online coupling in time* $\mathrm{poly}(nk)$ *with p-cost*[3]
$$\mathsf{T}_{\mathsf{c}^p}^{1/p}(A_k^{\hat{\mu},\hat{\nu}}) \le \delta + \Delta$$
in which $\delta = 2\left(\sum_{i\in[n]} \mathsf{T}^{\mathrm{Em}}_{\mathsf{c}_i,k}(\mu_i)\right)^{1/p}$ *and* $\Delta = \Delta_{\mathsf{c}^p}^{1/p}(\mu,\nu)$ *as in Definition 9.*[4]
2. *The average total cost of A calling* $\hat{\mu},\hat{\nu}$ *is as follows.* $\mathsf{sc}_\nu^A \le k\cdot\mathsf{sc}_\nu$ *and* $\mathsf{sc}_\mu^A \le k\cdot\mathsf{sc}_\mu$.[5]
3. *There is an algorithm B that achieves the same as A does, but it transports* ν *back to* μ.

Proof. At a high level, we use an empirical variant of the greedy algorithm (which is related to the KR transport) to design the algorithm. The algorithm itself is quite simple; the bulk of the work goes into its analysis, which is quite delicate and uses many tools from Section 3.

The Transportation Algorithm A. The algorithm A works in n rounds. In round $i\in[n]$, given $x_i\sim\mu_i$ find $y_i\sim\nu_i|y_{[i-1]}$ as described below.

1. For $j\in[k]$, let $y_i^{(j)}\sim\hat{\nu}(y_{[i-1]})$ be independent samples forming the multi-set $\mathcal{Y}$ of size k.
2. Pick $t\leftarrow[k]$ at random. For all $j\in[k], j\neq t$, let $x_i^{(j)}\sim\mu$ be $k-1$ independent samples. Additionally, let $x_i^{(t)}=x_i$, and $\mathcal{X}$ be the multi-set $\left\{x_i^{(j)} \mid j\in[k]\right\}$ of size k.
3. Find the optimal transport between the two distributions $U_{\mathcal{X}}, U_{\mathcal{Y}}$ under the cost c_i (e.g., using the Hungarian method[6]) that is in the form of a matching between $\mathcal{X}$ and $\mathcal{Y}$.[7]
4. Output $y_i\in\mathcal{Y}$ that is matched with $x_i^{(t)}=x_i\in\mathcal{X}$.

We now analyze the algorithm A above.

Transportation. A's running time is clearly $\mathrm{poly}(kn)$. We now prove that A's algorithm produces an *online* coupling between μ,ν, by showing that in round i, it couples μ_i and $\nu_i|y_{[i-1]}$. It is simple to check that all the elements of $\mathcal{X}$ are distributed as μ_i and all the elements of $\mathcal{Y}$ are distributed as in $\nu_i|y_{[i-1]}$. At first, it might not be clear why y_i is distributed as $\nu_i|y_{[i-1]}$, because the matching algorithm might change its distribution by picking it adversarially. However, since the algorithm hides the index of x_i and statistically hides it among $\mathcal{X}$, the final "matched pair" (x_i,y_i) is a random edge of the optimal matching/transport. Therefore, y_i is also distributed accurately, and hence A is producing an online coupling.

More formally, we can choose $t\in[k]$ at random *after* the matching between $\mathcal{X},\mathcal{Y}$ is chosen. Moreover, the marginal distribution of $y_i^{(j)}$ is $\hat{\nu}(y_{[i-1]})$. Therefore, for every (even fixed) matching between $\mathcal{X},\mathcal{Y}$, picking t at random will lead to picking $y_i=y_i^{(j)}$ where j is the index of the sample in $\mathcal{Y}$ that is matched with the index t in $\mathcal{Y}$. Therefore, $y_i\sim\hat{\nu}(y_{[i-1]})$.

The Cost. To analyze the transportation cost we apply Lemma 23 from Section 3, which is stated in a more general form to better demonstrating the key ideas.

To apply Lemma 23, let $J(y_{[i-1]})$ return pair of distributions $(\mu_i'=U_{\mathcal{X}},\nu_i'=U_{\mathcal{Y}})$ that are constructed using independent sample multi-sets $\mathcal{X},\mathcal{Y}$ of size k, in order, from $\mu_i,\nu_i|y_{[i-1]}$. Finally, because the algorithm A finds an optimal transport between μ_i',ν_i', we will have the premises of Lemma 23 and conclude that

$$\mathsf{T}_{\mathsf{c}^p}^{1/p}(A_k^{\hat{\mu},\hat{\nu}}) \le \left(\mathop{\mathbb{E}}_{(x,y)\sim\pi}\sum_{i\in[n]}\mathop{\mathbb{E}}_{(U_{\mathcal{X}},U_{\mathcal{Y}})\sim J(y_{[i-1]})}\mathsf{T}_{\mathsf{c}_i}\left(U_{\mathcal{X}},\sigma_{z_{[i-1]}}^{-1}\sharp U_{\mathcal{Y}}\right)\right)^{1/p} + \Lambda_{\mathsf{c}^p}^{1/p}(\pi_A),$$

[3] See Definition 1.
[4] By Theorem 10, Δ_{c^p} is also equal to $\mathsf{T}^{\mathrm{OnT}}_{\mathsf{c}^p}(\mu,\nu)=\mathsf{T}^{\mathrm{OnC}}_{\mathsf{c}^p}(\mu,\nu)=\mathsf{T}^{\mathrm{G}}_{\mathsf{c}^p}(\mu,\nu)$.
[5] Note that because μ is a product distribution, if sc_μ models the computational cost of sampling from μ, then we would have $\mathsf{sc}_\mu=\sum_{i\in[n]}\mathsf{sc}_{\mu_i}$, where sc_{μ_i} models the computational cost of sampling from μ_i.
[6] This method can be implemented faster when the cost function is convex, in which case simply sorting $\mathcal{X},\mathcal{Y}$ gives us the optimal matching, as a monotone mapping.
[7] This can be proved, e.g., using the Birkhoff–von Neumann decomposition of doubly stochastic matrices.

in which $\sigma_{z_{[i-1]}}$ is an (optimal) transport from μ_i to $\nu_i|y_{[i-1]}$. (See Definition 21 for the $\sharp$ notation.) We now further simplify the summation above.

- Because $U_{\mathcal{X}}, U_{\mathcal{Y}}$ are empirical distributions from $\mu_i, \nu_i|y_{[i-1]}$, if we let $U_{\mathcal{X}} = \mu_i', U_{\mathcal{Y}} = \nu_i'$ in Proposition 22, by Part 3 we get $U_{\mathcal{X}'} = \sigma^{-1}_{z_{[i-1]}} \sharp U_{\mathcal{Y}}$ (see Definition 21 for the notation) in which $U_{\mathcal{X}'}$ is also an empirical distribution sampled from μ_i independently of $U_{\mathcal{X}}$. So, the first term of the right hand side in the inequality above simplifies to:

$$\left(\mathop{\mathbb{E}}_{(x,y)\sim\pi} \sum_{i\in[n]} \mathop{\mathbb{E}}_{\mathcal{X},\mathcal{X}'\sim\mu_i^k} \mathsf{T}_{\mathsf{c}_i}(U_{\mathcal{X}}, U_{\mathcal{X}'}) \right)^{1/p}$$

- Now, in the first term, both coordinates of $(x, y) \sim \pi$ are irrelevant to the summation.
- Since A is producing an *online* coupling the second term simplifies into $\Delta^{1/p}_{\mathsf{c}^p}(\mu, \nu) = \Lambda^{1/p}_{\mathsf{c}^p}(\pi_A)$, due to Part 3 of Proposition 12 and that μ is a product.
- Finally, by the triangle inequality of Proposition 16, the first term will become at most

$$2\left(\sum_{i\in[n]} \mathsf{T}^{\mathrm{Em}}_{k,\mathsf{c}_i}(\mu_i) \right)^{1/p} = 2\delta.$$

To apply Proposition 16, we let J_i to be the distribution over distributions that outputs the following triple of distributions (ν_1, ν_2, ν_3), where

$$\nu_1 = U_{\mathcal{X}}, \mathcal{X} \sim \mu_i^k, \nu_2 = \mu_i, \nu_3 = U_{\mathcal{X}'}, \mathcal{X}' \sim \mu_i^k.$$

Oracle Costs. In each round, A asks $k-1$ samples from μ_i and k samples from $\nu_i|y_{[i-1]}$. Furthermore, the previous samples $y_{[i-1]}$ are sampled according to $\nu_{[i-1]}$ itself, so the average total cost will be as claimed.

Inverse Transport. The reverse mapping uses the same algorithm for one dimension transport, but it maps $\nu_i|y_{[i-1]}$ to μ_i, and inspection shows its transportation and (expected) total oracle costs will be the same as that of A. ◀

4.1 Extending Transport to Conditional Distributions

In this subsection, we study how to use the main result of Theorem 28 and obtain transports from the same μ to a more rich set of distributions that can be obtained from ν by conditioning ν on an event $\mathcal{S}$ of not-so-small probability. Doing so would be extremely useful, when later on, we focus on transporting Gaussian distributions to the same distributions conditioned on an event $\mathcal{S}$. To prove this extension, we prove a general result about using sequential samplers for ν to obtain sequential samplers for $\nu|\mathcal{S}$.

▶ **Theorem 29** (Sequential Samplers for Event-conditioned Distributions)**.** *Suppose ν is an n-dimensional distribution that has a sequential sampler $\hat{\nu}$ with average cost sc_ν. Suppose $\mathcal{S}$ is an event of measure $\nu(\mathcal{S}) \geq \varepsilon$, and $\omega = \nu|\mathcal{S}$ is ν conditioned on $\mathcal{S}$. Then, there is an algorithm O that uses oracle $\hat{\nu}$ and a membership oracle $\mathcal{S}$ and achieves the following.*

1. *For all $y_{[i-1]} \sim \omega_{[i-1]}$, $O^{\mathcal{S},\hat{\nu}}(y_{[i-1]}) \sim \hat{\omega}(y_{[i-1]})$.*
2. *If we define $\mathsf{sc}_\omega(y_{[i-1]})$ be the average total cost of $O^{\mathcal{S},\hat{\nu}}(y_{[i-1]})$ querying $\hat{\nu}$, and if we define $\mathsf{sc}_\nu(i) = \mathbb{E}_{y\sim\nu}\, \mathsf{sc}_\nu(y_{[i-1]})$, then*

$$\mathsf{sc}_\omega \leq \frac{1}{\varepsilon} \sum_{i\in[n]} i \cdot \mathsf{sc}_\nu(i) \leq n \cdot \frac{\mathsf{sc}_\nu}{\varepsilon}.$$

3. *When iteratively sampling* $(y_1, \ldots, y_n) \sim \omega$*, the expected number of calls to* $\mathcal{S}$ *in round* i *is at most* $1/\varepsilon$*, making the total expected number of calls to* $\mathcal{S}$ *to be at most* n/ε*.*
4. *The running time of the iterative sampling of* $(y_1, \ldots, y_n) \sim \omega$*, relative to the provided oracles* $\hat{\nu}, \mathcal{S}$ *is at most* $O(n^2/\varepsilon)$*.*

In other words, one can use $O^{\mathcal{S},\hat{\nu}}$ to emulate a sequential sampler for $\omega = \nu|\mathcal{S}$ in such a way that the average cost of obtaining a full sequence $y \sim \nu|\mathcal{S}$ using n nested calls to the provided sequential sampler only goes up (at most) by a multiplicative factor $n/\Pr[\mathcal{S}]$.

The main idea in the proof is to use rejection sampling with a subtle analysis. Namely, $O^{\mathcal{S},\hat{\nu}}$ simply keeps using $\hat{\nu}$ to obtain full sequences multiple times until the sample sequence falls within the event $\mathcal{S}$. The full proof follows.

Proof of Theorem 29. For $v = (v_1, \ldots, v_n)$, let $v_{\geq i} = (v_i, \ldots, v_n)$ and $v = (v_{[i-1]}, v_{\geq i})$.

Our algorithm $O^{\mathcal{S},\hat{\nu}}(y_{[i-1]})$ samples from $\hat{\omega}(y_{[i-1]})$ as follows.

1. Sample from $\nu|y_{[i-1]}$ as follows: for $j = i, \ldots, n$ sample fresh values $y_j \sim \hat{\nu}(y_{[j-1]})$.
2. If $y = (y_{[i-1]}, y_{\geq i}) \in \mathcal{S}$, then output y_i; otherwise, go back to the previous step.

We refer to each execution of the two steps above (that has exactly one call to $\mathcal{S}$) a *trial*.

Part 1 follows from the fact that the above sampling process is a simple rejection sampling. To prove Part 2, let $H(y_{[i-1]})$ be a random variable that counts the number of trials, and let its expectation be

$$h(y_{[i-1]}) = \mathbb{E}[H(y_{[i-1]})] = \frac{1}{\Pr_{y\sim\nu|y_{[i-1]}}[y \in \mathcal{S}]}.$$

Also let $\overline{\mathsf{sc}}_\nu(y_{[i-1]}) = \mathbb{E}_{y\sim\nu|y_{[i-1]}} \sum_{j\geq i} \mathsf{sc}_\nu(y_{j-1})$. It can be observed that $\mathbb{E}_{y\sim\nu} \overline{\mathsf{sc}}_\nu(y_{[i-1]}) = \sum_{j\geq i} \mathsf{sc}_\nu(i)$. Using these notations, the oracle sampling cost of $\hat{\omega}(\cdot)$ at $y_{[i-1]}$ will be

$$\mathsf{sc}_\omega(y_{[i-1]}) = h(y_{[i-1]}) \cdot \overline{\mathsf{sc}}_\nu(y_{[i-1]}).$$

Therefore, the average cost of $\hat{\omega}$ will be

$$\mathsf{sc}_\omega = \mathop{\mathbb{E}}_{y\sim\omega} \sum_{i\in[n]} h(y_{[i-1]}) \cdot \mathsf{sc}_\omega(y_{[i-1]}) = \sum_{i\in[n]} \mathop{\mathbb{E}}_{y\sim\omega} h(y_{[i-1]}) \cdot \mathsf{sc}_\omega(y_{[i-1]}).$$

A subtle point is that, in the above sums the first half $y_{[i-1]}$ is sampled conditioned on $\mathcal{S}$, while the second half is done without such conditioning. We claim that for each i, we have

$$\mathop{\mathbb{E}}_{y\sim\omega} h(y_{[i-1]}) \cdot \mathsf{sc}_\omega(y_{[i-1]}) \leq \frac{1}{\varepsilon} \cdot \mathop{\mathbb{E}}_{y\sim\nu} \mathsf{sc}_\omega(y_{[i-1]}). \tag{1}$$

Note that if Eq. (1) holds, then we conclude Part 2, because we get:

$$\mathsf{sc}_\omega \leq \sum_{i\in[n]} \frac{1}{\varepsilon} \mathop{\mathbb{E}}_{y\sim\nu} \mathsf{sc}_\omega(y_{[i-1]}) = \frac{1}{\varepsilon} \cdot \sum_{i\in[n]} \sum_{j\geq i} \mathsf{sc}_\nu(i) = \frac{1}{\varepsilon} \sum_{i\in[n]} i \cdot \mathsf{sc}_\nu(i).$$

The following lemma proves Eq. (1) using $\mathcal{U} = \mathrm{Supp}(\nu_{[i-1]}), \mathcal{V} = \mathrm{Supp}(\nu_{\geq i})$, $\sigma = \nu$, $f(y) = \overline{\mathsf{sc}}_\nu(y_{[i-1]})$ and $\mathcal{S}$ as before.

▶ **Lemma 30** (Expected Cost of Two-Step Sequential Sampling)**.** *Suppose* σ *is distributed over* $\mathcal{U} \times \mathcal{V}$ *with margins* $\sigma_\mathcal{U}, \sigma_\mathcal{V}$*, and* $\mathcal{S} \subseteq \mathcal{U} \times \mathcal{V}$ *has probability* $\sigma(\mathcal{S}) = \varepsilon$*. Also, suppose* f *is a random variable defined over* σ *with average* $\bar{f}$*. Consider the following process: (1) Sample* $u \sim \sigma_\mathcal{U}|\mathcal{S}$*, which is the marginal distribution of* $\mathcal{U}$ *in* $\sigma|\mathcal{S}$ *and let* $\varepsilon_u = \Pr_{v\sim\sigma^u_\mathcal{V}}[(u, v) \in \mathcal{S}]$*, in which* $\sigma^u_\mathcal{V}$ *is the marginal distribution over* $\mathcal{V}$ *in* σ *conditioned on* $\sigma_\mathcal{U} = u$*. (2) Sample* $v \sim \sigma^u_\mathcal{V}|\mathcal{S}$*. Then,*

$$\mathop{\mathbb{E}}_{(u,v)} \frac{f(u, v)}{\varepsilon_u} \leq \frac{\bar{f}}{\varepsilon}.$$

Proof. We write the proof for the discrete setting. A similar proof holds in general. For each $u \sim \sigma_{\mathcal{U}}$, define $p_u = \Pr[\sigma_{\mathcal{U}} = u]$ and $f_u = \mathbb{E}_{v\sim\sigma_{\mathcal{V}}|u} f(u,v)$. We have $\varepsilon = \sum_{u\in\mathcal{U}} p_u\varepsilon_u$, and $q_u = \frac{p_u\cdot\varepsilon_u}{\varepsilon}$ is the probability we sample u in the sampling process of the lemma statement. Then, if we let $\mathcal{U}_{\mathcal{S}} = \{u \mid \varepsilon_u > 0\} = \mathrm{Supp}(\sigma_{\mathcal{U}}|\mathcal{S})$, we have

$$\mathop{\mathbb{E}}_{u} \frac{1}{\varepsilon_u} \mathop{\mathbb{E}}_{v|u} f(u,v) = \sum_{u\in\mathcal{U}_{\mathcal{S}}} \frac{q_u}{\varepsilon_u} f_u = \sum_{u\in\mathcal{U}} \frac{p_u}{\varepsilon} f_u \le \sum_{u\in\mathcal{U}_{\mathcal{S}}} \frac{p_u}{\varepsilon} f_u = \frac{\bar{f}}{\varepsilon}. \qquad \blacktriangleleft$$

To prove Part 3, using Lemma 30 and $f(u,v) = 1$, we conclude that the expected number of times we call the $\mathcal{S}$ oracle at each node $y_{[i-1]}$ is at most $1/\varepsilon$.

To prove Part 4 we can simply use a fake oracle sampling cost of $\hat{\mathsf{sc}}'_\nu(\cdot) = 1$. Then the claim about the running time follows from Part 2. $\blacktriangleleft$

Deriving corollaries. Using Theorem 29, we can derive more transportation results from Theorem 28 by conditioning ν on an arbitrary event $\mathcal{S}$ for which we have a membership oracle at hand. Note that the parameter Δ will change to a new value, but the key point is that we can control the cost of sequential samples from the new oracle, so long as we could do so for the initial oracle. Another interesting application of Theorem 29 is to transport a product distribution μ to $\mu|\mathcal{S}$ for an arbitrary event $\mathcal{S}$, obtaining the following corollary.[8]

▶ **Corollary 31.** *Suppose the assumptions of Theorem 28 hold. Then, we have the following:*

1. *There is an algorithm M_k such that, for all events $\mathcal{S}$ defined over μ, $M_k^{\mathcal{S},\hat{\mu}}$ transports μ to $\mu|\mathcal{S}$ in expected time* $\mathrm{poly}(nk/\varepsilon)$ *and p-cost $\mathsf{T}_{\mathsf{c}^p}^{1/p}(M_k^{\hat{\mu},\hat{\nu}}) \le \delta + \Delta$, in which δ is as in Theorem 28 and $\Delta = \Delta_{\mathsf{c}^p}^{1/p}(\mu,\mu|\mathcal{S})$.*
2. *There is an algorithm N_k such that, for all events $\mathcal{S}$ defined over ν, $N_k^{\mathcal{S},\hat{\mu},\hat{\nu}}$ transports μ to $\nu|\mathcal{S}$ in expected time* $\mathrm{poly}(nk/\varepsilon)$ *and p-cost $\mathsf{T}_{\mathsf{c}^p}^{1/p}(N_k^{\mathcal{S},\hat{\mu},\hat{\nu}}) \le \delta + \Delta$, in which δ is as in Theorem 28 and $\Delta = \Delta_{\mathsf{c}^p}^{1/p}(\mu,\nu|\mathcal{S})$. Moreover, $\mathsf{sc}_\nu^N \le n\cdot\mathsf{sc}_\nu^A/\varepsilon$, for A of Theorem 28.*

In both cases above, the expected number of calls to $\mathcal{S}$ is at most kn/ε, and the transportation can be reversed with the same upper bounds on the running time and oracle costs.

5 Algorithmic Transport for Gaussian

In this section we focus on cases where at least one of the two distributions involved in the transport is Gaussian. We first use the main result of Theorem 28 and derive an algorithmic variant of Talagrand's result [43] about transporting Gaussian measure to arbitrary distributions with bounded KL divergence from Gaussian. We then derive, as a corollary, a computational concentration result for the Gaussian source measure under the ℓ_2 distance. Finally, we focus on finding (optimal) online transports in cases where both the source and destination are Gaussians, but they could be arbitrary (non-product) Gaussians.

5.1 Algorithmic Variant of Talagrand's Transport for Gaussian

▶ **Theorem 32** (Algorithmic Version of Talagrand's Gaussian Transport Theorem)**.** *Let Φ^n be the standard Gaussian in dimension n and ν be an arbitrary distribution in $\mathbb{R}^n$. There is an algorithm A_k, with integer parameter k, such that whenever $A_k^{\hat{\nu}}$ is provided with a sequential sampler $\hat{\nu}$ for ν, the following properties hold.*

[8] In the next section we apply this idea to the special case of Gaussian distributions.

1. *For all $p \geq 1$ and ν, $A_k^{\hat{\nu}}$ transports Φ^n to ν in time $O(nk \log k)$ with p-cost at most*

$$\mathsf{T}_{\ell_p^p}^{1/p}(A_k^{\hat{\nu}}) \leq \Delta_{\ell_p^p}^{1/p}(\Phi^n, \nu) + \left(O_p(nk^{-1/2})\right)^{1/p}.$$

   For $p = 2$, by the Talagrand inequality of Theorem 24, we have $\Delta_{\ell_2^2}(\Phi^n, \nu) \leq 2\mathsf{KL}(\Phi^n, \nu)$.
2. *The average total oracle cost of $A_k^{\hat{\nu}}$ is at most $k \cdot \mathbb{E}_{y\sim\nu} \sum_{i\in[n]} \mathsf{sc}(\nu_i | y_{[i-1]})$.*
3. *There is an algorithm $B_k^{\hat{\nu}}$ that achieves the same as $A_k^{\hat{\nu}}$, but it transports ν back to Φ^n.*

▶ Remark 33 (Working with ℓ_p instead of ℓ_p^p). One might wonder what happens if we want to measure (and upper bound) transfer costs using ℓ_p rather than ℓ_p^p. However, this can be obtained using Jensen's inequality (or rather the monotonicity of Wasserstein p-costs for a fixed cost c). In particular, for every coupling π, we have $\mathsf{T}_{\ell_p}(\pi) \leq \mathsf{T}_{\ell_p^p}^{1/p}(\pi)$ for all $p \geq 1$. Hence Theorem 32 is stated in the stronger form already.

Proof of Theorem 32. The proof follows directly from Theorem 28 and Lemma 26. Namely, we use Corollary 26 to bound the term δ in Theorem 28 that upper bounds the transportation cost of empirical Gaussian from the Gaussian itself. One small point here is that, we will not need oracle samplers from the Gaussian itself, as we can use well-known sampling methods such as the Box-Muller method that generate such samples efficiently [36].[9] ◀

We now focus on a special case of interest, in which the target distribution ν is $\Phi^n|\mathcal{S}$ for an event $\mathcal{S}$ of probability $\Phi^n(\mathcal{S}) = \varepsilon$, and show that in this case, one can have a single online transportation algorithm that uniformly works for all $\mathcal{S}$ by merely accessing $\mathcal{S}$ through a membership oracle. We first define such uniform transportation algorithms.

▶ **Definition 34** (Oracle Set-Transport). *For distribution μ and transportation cost c, we say that (μ, c) has a* set-transport *of cost at most $\kappa(\cdot)$ for a non-increasing function $\kappa\colon [0,1] \mapsto [0,1]$, if for every event $\mathcal{S} \subseteq \mathrm{Supp}(\mu)$, it holds that $\mathsf{T}_{\mathsf{c}}(\mu, \mu|\mathcal{S}) \leq \kappa(\mu(\mathcal{S}))$. We further say that (μ, c) has an* oracle *set-transport of cost at most $\kappa(\cdot)$ if there is a single algorithm A such that with oracle membership queries for an arbitrary set $\mathcal{S}$ and sampling queries for μ, $A^{\mathcal{S},\mu}$ produces a transport of cost at most $\kappa(\mu(\mathcal{S}))$ from μ to $\mu|\mathcal{S}$.*

▶ **Theorem 35** (Oracle-Set Transport for Gaussian Measure). *Let Φ^n be the standard Gaussian in dimension n. There is an (online) oracle-set transport algorithm A_k for Φ^n such that:*
1. *For all $p \in [1,2]$ and $\mathcal{S}$ of measure $\Phi^n(\mathcal{S}) = \varepsilon$,*

$$\mathsf{T}_{\ell_p^p}^{1/p}(A_k^{\mathcal{S}}) \leq \kappa^{1/p}(\varepsilon) = n^{1/p-1/2}\sqrt{2\ln 1/\varepsilon} + \left(O_p(nk^{-1/2})\right)^{1/p},$$

   which is at most $(1+\gamma) \cdot n^{1/p-1/2}\sqrt{2\ln 1/\varepsilon}$, for sufficiently large $k = \mathrm{poly}(n, 1/\varepsilon, 1/\gamma)$.
2. *In expectation, $A_k^{\mathcal{S}}$ asks at most kn/ε queries to $\mathcal{S}$ and runs in $\mathrm{poly}(nk/\varepsilon)$.*
3. *There is an algorithm B_k that achieves the same, but $B_k^{\mathcal{S}}$ transports $\Phi^n|\mathcal{S}$ back to Φ^n.*

Proof of Theorem 35. To prove Theorem 35 we first use the first item of Corollary 31 where $\mu = \Phi^n$. This way, we already know that the running time of the transportation algorithm and its number of calls to $\mathcal{S}$ are bounded as stated.

Then, we need to bound both terms Δ, δ. To bound δ, we again use Corollary 26 as we did in the proof of Theorem 32. To bound Δ, we again use Corollary 24 and the well-known fact that $\mathsf{KL}(\mu|\mathcal{S}, \mu) \leq \ln 1/\varepsilon$ for $\mathcal{S}$ such that $\mu(\mathcal{S}) \geq \varepsilon$ (applied to $\mu = \Phi^n$). ◀

[9] In particular, given two independent and uniform $u_1, u_2 \sim [0,1]$, the sampling works as follows: $v_1 = \sqrt{-2\ln u_1}\cos(2\pi u_2), v_2 = \sqrt{-2\ln u_1}\sin(2\pi u_2)$ are independent samples $v_1, v_2 \sim \mathcal{N}(0,1)$.

Due to our transports being "reversible", one can obtain a variant of the result above that transports conditional distributions to conditional distributions through composition.

5.2 Dimension-Independent Computational Concentration for Gaussian

It is well-known that transportation inequalities can be used to derive concentration of measure results [22]. Recently, a *computational* variant of this phenomenon has been explored [31, 15], which bears similarities to how we make transportation algorithmic. In a computational concentration result, we need an algorithm that maps "most" of the sampled points from the space to any "sufficiently large" event $\mathcal{S}$, algorithmically. The "cost" of the concentration is (a worst-case) allowed distance d that the algorithm is allowed to move the points, and its error is the fraction of the sampled points that it fails to map to $\mathcal{S}$ withing the allowed distance d. The work of [15] obtained such results optimally for some settings (e.g., Gaussian under ℓ_1 distance), however they left open obtaining an optimal (dimension-free) computational concentration result for the Gaussian space under the ℓ_2 distance.

Using Theorem 35, we can resolve the question left open in [15] and derive such optimal computational concentration for the Gaussian space under ℓ_2 as a simple corollary to our algorithmic transport result. Theorem 36 below follows from Theorem 35 and the Markov inequality. Using $p = 2$ below implies the desired dimension-independent result.

▶ **Corollary 36** (Computational Concentration for Gaussian). *For all $\varepsilon, \delta, \lambda, p \in [1, 2]$, given oracle access to $\mathcal{S} \subseteq \mathbb{R}^n$, $A_k^{\mathcal{S}}(x)$ of Theorem 35 runs in $\mathrm{poly}(\frac{n}{\varepsilon\lambda})$-time and with probability $1 - \delta$ over $x \sim \Phi^n$, it finds a point $y \in \mathcal{S}$ of distance*

$$\ell_p(x, y) \leq \frac{(1+\lambda) \cdot n^{1/p-1/2} \sqrt{2 \ln 1/\varepsilon}}{\delta}.$$

References

1 Luigi Ambrosio, Nicola Gigli, and Giuseppe Savaré. *Gradient flows: in metric spaces and in the space of probability measures*. Springer Science & Business Media, 2008.

2 Martín Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International Conference on Machine Learning*, 2017. URL: `http://proceedings.mlr.press/v70/arjovsky17a/arjovsky17a.pdf`.

3 Julio Backhoff, Mathias Beiglbock, Yiqing Lin, and Anastasiia Zalashko. Causal transport in discrete time and applications. *SIAM Journal on Optimization*, 27(4):2528–2562, 2017.

4 Arjun Nitin Bhagoji, Daniel Cullina, and Prateek Mittal. Lower bounds on adversarial robustness from optimal transport. *Advances in Neural Information Processing Systems*, 32, 2019.

5 Arnab Bhattacharyya, Sutanu Gayen, Kuldeep S Meel, Dimitrios Myrisiotis, A Pavan, and NV Vinodchandran. On approximating total variation distance. In *IJCAI*, 2023.

6 Jeremiah Birrell and Mohammadreza Ebrahimi. Adversarially robust deep learning with optimal-transport-regularized divergences. *arXiv preprint*, 2023. `doi:10.48550/arXiv.2309.03791`.

7 Jose Blanchet, Karthyek Murthy, and Fan Zhang. Optimal transport-based distributionally robust optimization: Structural properties and iterative schemes. *Mathematics of Operations Research*, 47(2):1500–1529, 2022. `doi:10.1287/moor.2021.1178`.

8 Lenore Blum. *Complexity and real computation*. Springer Science & Business Media, 1998.

9 Mark Braverman. On the complexity of real functions. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*, pages 155–164. IEEE, 2005. `doi:10.1109/SFCS.2005.58`.

10 Yann Brenier. Polar factorization and monotone rearrangement of vector-valued functions. *Communications on pure and applied mathematics*, 44(4):375–417, 1991.

11 Maarten Buyl and Tijl De Bie. Optimal transport of classifiers to fairness. In *Advances in Neural Information Processing Systems*, 2022.

12 Guillaume Carlier, Alfred Galichon, and Filippo Santambrogio. From knothe's transport to brenier's map and a continuation method for optimal transport. *SIAM Journal on Mathematical Analysis*, 41(6):2554–2576, 2010. doi:10.1137/080740647.

13 Sinho Chewi, Jonathan Niles-Weed, and Philippe Rigollet. Statistical optimal transport. *arXiv preprint*, 2024. arXiv:2407.18163.

14 Nicolas Courty, Rémi Flamary, Amaury Habrard, and Alain Rakotomamonjy. Joint distribution optimal transportation for domain adaptation. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 3733–3742, 2017.

15 Omid Etesami, Saeed Mahloujifar, and Mohammad Mahmoody. Computational concentration of measure: Optimal bounds, reductions, and more. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 345–363. SIAM, 2020. doi:10.1137/1.9781611975994.21.

16 Weiming Feng, Heng Guo, Mark Jerrum, and Jiaheng Wang. A simple polynomial-time approximation algorithm for the total variation distance between two product distributions. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 343–347. SIAM, 2023. doi:10.1137/1.9781611977585.CH30.

17 Alessio Figalli and Cédric Villani. *Optimal Transport and Curvature*, pages 171–217. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. doi:10.1007/978-3-642-21861-3_4.

18 Nicolas Ford, Daniel Duckworth, Mohammad Norouzi, and George E Dahl. The importance of generation order in language modeling. *arXiv preprint*, 2018. arXiv:1808.07910.

19 Nicolas Fournier and Arnaud Guillin. On the rate of convergence in wasserstein distance of the empirical measure. *Probability Theory and Related Fields*, 162(3):707–738, 2015.

20 Charlie Frogner, Chiyuan Zhang, Hossein Mobahi, Mauricio Araya, and Tomaso A. Poggio. Learning with a wasserstein loss. In *Advances in Neural Information Processing Systems*, volume 28, pages 2053–2061, 2015.

21 Alfred Galichon. The unreasonable effectiveness of optimal transport in economics. *Proceeding of the 2020 World Congress of the Econometric Society*, 2020.

22 Nathael Gozlan and Christian Léonard. Transport inequalities. a survey. *Markov Processes And Related Fields*, 16:635–736, 2010.

23 Steven Haker, Lei Zhu, Allen Tannenbaum, and Sigurd Angenent. Optimal mass transport for registration and warping. *International Journal of computer vision*, 60(3):225–240, 2004. doi:10.1023/B:VISI.0000036836.66311.97.

24 Jan-Christian Hütter and Philippe Rigollet. Minimax estimation of smooth optimal transport maps. *The Annals of Statistics*, 49(2), 2021.

25 Leonid V Kantorovich. On the translocation of masses. In *Dokl. Akad. Nauk. USSR (NS)*, volume 37, pages 199–201, 1942.

26 Daegyu Kim et al. Improving diffusion-based generative models via approximated optimal transport. *arXiv preprint*, 2024. arXiv:2403.05069.

27 Herbert Knothe. Contributions to the theory of convex bodies. *Michigan Mathematical Journal*, 4(1):39–52, 1957.

28 Martin Knott and Cyril S Smith. On the optimal mapping of distributions. *Journal of Optimization Theory and Applications*, 43:39–49, 1984.

29 Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

30 Rémi Lassalle. Causal transport plans and their monge–kantorovich problems. *Stochastic Analysis and Applications*, 36(3):452–484, 2018.

31 Saeed Mahloujifar and Mohammad Mahmoody. Can adversarially robust learning leverage computational hardness? In *Algorithmic Learning Theory*, pages 581–609. PMLR, 2019. URL: `http://proceedings.mlr.press/v98/mahloujifar19a.html`.

32 Tudor Manole, Sivaraman Balakrishnan, Jonathan Niles-Weed, and Larry Wasserman. Plugin estimation of smooth optimal transport maps. *The Annals of Statistics*, 52(3):966–998, 2024.

33 Gaspard Monge. Mémoire sur la théorie des déblais et des remblais. *Histoire de l'Académie Royale des Sciences de Paris*, 1781.

34 Paul Montesuma, Loic Ngolè Mboula, and Antoine Souloumiac. Recent advances in optimal transport for machine learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

35 Ludovic Métivier, Romain Brossier, Jean Virieux, and Jesus de la Puente. Measuring the misfit between seismograms using an optimal transport distance. *Geophysical Journal International*, 205(1):345–377, 2016.

36 Raymond Edward Alan Christopher Paley and Norbert Wiener. *Fourier transforms in the complex domain*, volume 19. American Mathematical Soc., 1934.

37 Gabriel Peyré. Course notes on computational optimal transport. *Mathematical Tours*, 2024. URL: `https://mathematical-tours.github.io/`.

38 Gabriel Peyré, Marco Cuturi, et al. Computational optimal transport: With applications to data science. *Foundations and Trends® in Machine Learning*, 11(5-6):355–607, 2019.

39 Murray Rosenblatt. Remarks on a multivariate transformation. *The annals of mathematical statistics*, 23(3):470–472, 1952.

40 Filippo Santambrogio. Models and applications of optimal transport theory, 2009. Lecture Notes, Grenoble.

41 Shai Shalev-Shwartz et al. Online learning and online convex optimization. *Foundations and Trends® in Machine Learning*, 4(2):107–194, 2012.

42 Richard Sinkhorn and Paul Knopp. Concerning nonnegative matrices and doubly stochastic matrices. *Pacific Journal of Mathematics*, 21(2):343–348, 1967.

43 Michel Talagrand. Transportation cost for Gaussian and other product measures. *Geometric & Functional Analysis GAFA*, 6(3):587–600, 1996.

44 Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

45 Cédric Villani. *Topics in optimal transportation*, volume 58. American Mathematical Soc., 2021.

46 Hongkang Yang and Esteban G. Tabak. Clustering, factor discovery and optimal transport. *IMA Journal of Applied Mathematics*, 10(4):1353–1387, 2021. `doi:10.1093/imaiai/iaaa040`.

47 Yi-Zhuang You et al. Renormalization group flow, optimal transport and diffusion-based generative model. *Physical Review E*, 2024.

Finding Diverse Solutions in Combinatorial Problems with a Distributive Lattice Structure

Mark de Berg
Eindhoven University of Technology, The Netherlands

Andrés López Martínez
Eindhoven University of Technology, The Netherlands

Frits Spieksma
Eindhoven University of Technology, The Netherlands

Abstract

We generalize the polynomial-time solvability of k-Diverse Minimum s-t Cuts (De Berg et al., ISAAC'23) to a wider class of combinatorial problems whose solution sets have a distributive lattice structure. We identify three structural conditions that, when met by a problem, ensure that a k-sized multiset of maximally-diverse solutions – measured by the sum of pairwise Hamming distances – can be found in polynomial time. We apply this framework to obtain polynomial-time algorithms for finding diverse minimum s-t cuts, diverse stable matchings, and diverse market-clearing price vectors. Moreover, we show that the framework extends to two other natural measures of diversity. Lastly, we present a simpler algorithmic framework for finding a largest set of pairwise disjoint solutions in problems that meet these structural conditions.

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

Keywords and phrases Diversity, Lattice Theory, Submodular Function Minimization

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.11

Related Version *Full Version*: `https://arxiv.org/abs/2504.02369` [8]

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.

1 Introduction

In combinatorial optimization problems, the objective is typically to identify a single optimal solution. However, this approach may be inadequate or impractical in real-world situations, where some constraints and factors are often overlooked or unknown in advance. This motivates the development of algorithms capable of finding multiple solutions, with *diversity* playing a key role. A growing body of research has focused on finding diverse solutions in classical combinatorial problems, much of it emerging from the field of fixed-parameter tractability [1, 9, 12, 23, 24, 32, 38, 29]. These studies show that finding diverse solutions is, in general, computationally more challenging than finding a single one. For instance, while Maximum Matching is solvable in polynomial time, finding two edge-disjoint perfect (or maximum) matchings is NP-hard, even on 3-regular graphs [12].

In this paper, we aim to develop theoretically efficient algorithms that produce a collection of maximally diverse solutions. We use the sum of pairwise Hamming distances between solutions as our measure of diversity. In contrast with the aforementioned literature, we show that a broader class of diverse problems is computationally no harder than finding a single solution in polynomial time. Specifically, we generalize the polynomial-time solvability of k-Diverse Minimum s-t Cuts by De Berg et al. [7] to a class of combinatorial problems whose solution sets form a distributive lattice.

© Mark de Berg, Andrés López Martínez, and Frits Spieksma;
licensed under Creative Commons License CC-BY 4.0

We state our main result in terms of a unified general problem: Max-Sum k-Diverse Solutions. Let E be a finite set with n elements, and let $\Gamma \subseteq 2^E$ be a set of feasible solutions. For two feasible solutions $X, Y \in \Gamma$, the symmetric difference, or Hamming distance, between them is defined as $X \triangle Y = (X \setminus Y) \cup (Y \setminus X)$. Let $(X_1, X_2, \ldots, X_k)$ be a collection of k subsets of E. We consider the *pairwise-sum* diversity measure: $d_{\text{sum}}(X_1, X_2, \ldots, X_k) = \sum_{1 \le i < j \le k} |X_i \triangle X_j|$. We define Max-Sum k-Diverse Solutions as follows.

Max-Sum k-Diverse Solutions. *Given a finite set E of size n, an implicitly defined family Γ of subsets of E, referred to as feasible solutions, and a membership oracle for $\Gamma \subseteq 2^E$, find a k-multiset $C = (X_1, X_2, \ldots, X_k)$ with $X_1, X_2, \ldots, X_k \in \Gamma$, such that $d_{\text{sum}}(C)$ is maximum.*

Here, k is a fixed constant; i.e., k is not part of the input. Our main result is as follows.

▶ **Theorem 1.** *Max-Sum k-Diverse Solutions can be solved in polynomial time if the set of feasible solutions Γ satisfies the following three properties:*

1. *There is a relation $\le$ such that the poset $(E, \le)$ can be expressed as a disjoint union of r chains, and each feasible solution $X \in \Gamma$ contains exactly one element from each chain.*
2. *The set of feasible solutions with componentwise order defines a distributive lattice.*
3. *A compact representation of this lattice can be constructed in polynomial time.*

Similar to the approach of De Berg et al. [7], we achieve this result via a reduction to the *submodular function minimization* problem (SFM) on a distributive lattice, which is known to be solvable in polynomial time [20, 27, 36]. More precisely, we show that the pairwise-sum measure (reformulated as a minimization objective) is a submodular function on a distributive lattice of appropriately ordered k-sized collections of feasible solutions. By applying this result, in Section 4, we obtain polynomial-time algorithms for finding maximally diverse k-sized collections of stable matchings and market-clearing price vectors, while also reproducing the findings of De Berg et al. for minimum s-t cuts.

For simplicity, we present our results in terms of the d_{sum} measure. However, in Section 5 we will show that the framework extends to at least two other measures of diversity: the *coverage* (d_{cov}) and *absolute-difference* (d_{abs}) measures. Lastly, we consider the problem of finding a largest set of pairwise disjoint solutions in problems whose feasible solution set satisfies properties 1 and 2 of Theorem 1. In Section 6 we present an algorithm for this problem that bypasses the need for SFM.

Remark. Recently and independently from us, Iwamasa *et al.* [26] also presented a general framework for solving Max-Sum Diverse Solutions (and Max-Cov Diverse Solutions; see definition in Section 5) under certain conditions on the set of feasible solutions. They show that the conditions in our Theorem 1 imply the conditions in their framework. In the full version of this paper, we show that the reverse is true as well. Thus, their conditions and our conditions are, in fact, equivalent. Their approach reduces the problem to network flow, avoiding SFM and resulting in faster algorithms for computing diverse minimum s-t cuts and diverse stable matchings. However, in contrast to their framework, ours also supports the absolute-difference measure d_{abs} and thus, potentially applies to a broader range of problems.

2 Preliminaries

In this section, we introduce the notation and some basic results used throughout the paper. For a more comprehensive discussion on sets and posets, we refer to [25, 40], and for a detailed introduction to lattice theory, we refer to [2, 6, 19].

Sets, Multisets, and Tuples. For $k \in \mathbb{N}$, we use $[k]$ to denote the set $\{1, \ldots, k\}$. The power set of a set M is denoted by 2^M. For any set M, we use the symbol M^k for the cartesian product; $\{(a_1, a_2, \ldots, a_k) \mid a_i \in M\}$. The *disjoint union* of two sets is simply their union, but with the additional information that the two sets have no elements in common.

A *multiset* is a set in which elements can appear multiple times. The number of times an element appears in a multiset is referred to as its *multiplicity*. The *sum* of two multisets A and B, denoted by $A \uplus B$, is a multiset in which each element appears with a multiplicity equal to the sum of its multiplicity in A and in B. We refer to a multiset of cardinality k as a k-multiset. For a set M, we denote by M_k a k-multiset with elements drawn from M.

Unlike a multiset, where elements are unordered, a *tuple* is a collection of possibly repeated elements that is ordered. A k-tuple is a tuple of k elements. We denote a tuple by listing its elements within parenthesis and separated by commas; e.g., (a, b, c, d). Note that the cartesian product of k sets is a k-tuple.

Posets. A *partially ordered set (poset)* $P = (X, \preceq_P)$ consists of a ground set X along with a binary relation $\preceq_P$ on X that satisfies reflexivity, antisymmetry, and transitivity. When the relation $\preceq_P$ is evident from the context, we often use the same notation for both the poset and its ground set. In case a poset is indexed by a subscript i, we use $\preceq_i$ to denote its order relation. The Hasse diagram $G(P)$ of P, is a directed graph where each element of X is represented as a node, and an edge exists from element a to element b if $a \preceq_P b$ and no intermediate element c satisfies $a \preceq_P c \preceq_P b$. Vertices are arranged so that edge directions are implicitly understood as pointing upward.

A poset $P^* = (X^*, \preceq_P^*)$ is called a *subposet* of another poset $P = (X, \preceq_P)$ if (i) $X^* \subseteq X$ and (ii) for any $x, y \in X^*$ if $x \preceq_P^* y$ then $x \preceq_P y$. If the other direction of (ii) also holds, then we call P^* the subposet of P induced by X^*, and write $P^* = P[X^*]$. Given two posets $P = (X, \preceq_P)$ and $Q = (Y, \preceq_Q)$, their *disjoint union* $P \sqcup Q$ is the disjoint union of X and Y together with relation $\preceq_{P+Q}$ where $x \preceq_{P+Q} y$ if one of the following conditions holds: (i) $x, y \in X$ and $x \preceq_P y$, or (ii) $x, y \in Q$ and $x \preceq_Q y$. Thus, the Hasse diagram of $P \sqcup Q$ consists of the disconnected Hasse diagrams of P and Q drawn together.

A *chain* is a subset of a poset in which every pair of elements is comparable, and an *antichain* is a subset of a poset in which no two (distinct) elements are comparable. For any two elements x and y in a chain E with order relation $\preceq_E$, we say that x (resp. y) is a *chain-predecessor* (*chain-successor*) of y if $x \preceq_E y$. A poset is called a *chain decomposition* if the poset can be expressed as the disjoint union of chains.

For a poset $P = (X, \preceq_P)$, an *ideal* is a set $U \subseteq X$ where $u \in U$ implies that $v \in U$ for all $v \preceq_P u$. In terms of its Hasse diagram $G(P) = (X, E)$, a subset U of X is an ideal if and only if there is no incoming edge from U. We use $\mathcal{D}(P)$ to denote the family of all ideals of P. If $x \preceq_P y$ in the poset, then the closed interval from x to y, denoted by $[x, y]$, is the poset with ground set $\{z \in X \mid x \preceq_P z \preceq_P y\}$ together with relation $\preceq_P$.

Now we introduce the notion of *componentwise order*. Let $(X_i, \preceq_i)$, $i \in [r]$ be posets, with r a positive integer, and let $Y \subseteq X_1 \times \cdots \times X_r$. The componentwise order on Y is an order relation $\preceq$ defined as follows: Given two tuples $(a_1, a_2, \ldots, a_r)$ and $(b_1, b_2, \ldots, b_r) \in Y$, we write $(a_1, a_2, \ldots, a_r) \preceq (b_1, b_2, \ldots, b_r)$ iff $a_i \preceq_i b_i$ for all $i \in [r]$. Note that we drop the subscript in $\preceq$ whenever the order relation is a component-wise order. If the posets $(X_i, \preceq_i)$, $i \in [r]$, are all the same poset $(X, \preceq)$, we use $\preceq^r$ to denote the componentwise order on X^r and refer to it as the *product order*.

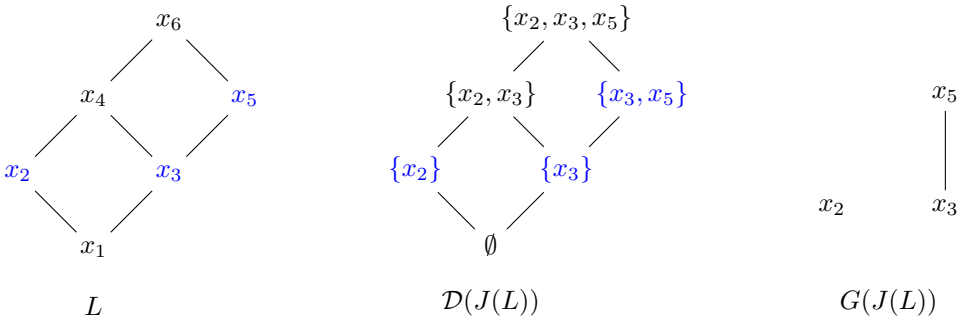


Figure 1 Example of Birkhoff's representation theorem for distributive lattices. The left is a distributive lattice L, the middle is the isomorphic lattice $\mathcal{D}(J(L))$ of ideals of join-irreducibles of L, and the right shows the compact representation $G(J(L))$ of L. The join irreducible elements of L and $\mathcal{D}(J(L))$ are highlighted in blue.

Lattices. 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 $x \vee y$. We can uniquely identify L by the tuple $(X, \vee, \wedge)$. The *bottom*, or minimum, element in the lattice L is denoted by $0_L := \bigwedge_{x \in L} x$. Likewise, the *top*, or maximum, element of L is given by $1_L := \bigvee_{x \in L} x$. 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. Every chain is a distributive lattice with max as join ($\vee$) and min as meet ($\wedge$).

Suppose we have a collection $L_1, \ldots, L_k$ of lattices $L_i = (X_i, \vee_i, \wedge_i)$ with $i \in [k]$. The *(direct) product lattice* $L_1 \times \ldots \times L_k$ is a lattice with ground set $X = \{(x_1, \ldots, x_k) : x_i \in L_i\}$ and join $\vee$ and meet $\wedge$ operations 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 k-th *power* of L. If L is a distributive lattice, then L^k is also distributive.

A crucial notion in this paper is that of *join-irreducibles*. An element x of a lattice L is called *join-irreducible* iff $x \neq 0_L$ and 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 – a fundamental tool for studying distributive lattices – 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. See Figure 1 for an illustration.

▶ **Theorem 2** (Birkhoff's Representation Theorem [2]). *Any distributive lattice L can be represented as the poset of its join-irreducibles $J(L)$, with the order induced from L.*

Hence, a distributive lattice L can be uniquely recovered from $J(L)$ and can thus be represented as the Hasse diagram of its poset of join-irreducibles, denoted by $G(J(L))$. We refer to $G(J(L))$ as a *compact representation* of L, since $J(L)$ is usually exponentially smaller than L. This representation is useful when designing algorithms, as the size of $G(J(L))$ is $O(|J(L)|^2)$, while L can have as many as $2^{|J(L)|}$ elements. Keep in mind, however, that Theorem 2 only guarantees the existence of such a compact representation; it does not provide a method to efficiently find the set $J(L)$.

Submodular Function Minimization. Let $f : X \to \mathbb{R}$ 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 $-f$. It is known that a submodular function on a distributive lattice L can be minimized in polynomial time in $|J(L)|$ – the number of join-irreducibles of L [20, 27, 36, 4].

▶ **Theorem 3** ([34, Note 10.15] and [30, 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.*

3 The Reduction to SFM

In this section, we prove Theorem 1 by reducing Max-Sum k-Diverse Solutions to SFM on a distributive lattice, under the assumption that the feasible solution set satisfies properties 1–3 of the theorem. The core ideas of this proof were already established by De Berg et al. [7] in the context of minimum s-t cuts. We include the argument here for completeness, while adapting and generalizing it to a broader setting.

The proof is divided into four parts, each supported by a corresponding lemma. The Distributivity Lemma (Lemma 4) shows that the set of (yet to be defined) *left-right ordered* k-tuples of feasible solutions, with product order, defines a distributive lattice L^*. The Cost-Equivalence Lemma (Lemma 7) further shows that optimizing the diversity over this lattice is the same as optimizing over the original set Γ_k of k-multisets of Γ. Hence, we can restrict ourselves to the elements of L^*. Next, the Submodularity Lemma (Lemma 11) establishes that the pairwise-sum measure (reformulated as a minimization objective) is a submodular function on L^*. Finally, the Compactness Lemma (Lemma 12) ensures that a compact representation of L^* can be constructed in polynomial time.

We begin by establishing some consequences of properties 1–3 of Theorem 1. Consider a ground set E and a set of feasible solutions $\Gamma \subseteq 2^E$ for which the properties hold. By property 1, we know that there is a poset associated to E that is the disjoint union of r chains $(E_i, \preceq_i)$, $i \in [r]$, and that each feasible solution $X \in \Gamma$ contains exactly one element from each chain, meaning $|X| = r$ and $\Gamma \subseteq E_1 \times \cdots \times E_r$. Then, the set Γ, with componentwise order $\preceq$, forms a poset of feasible solutions $L = (\Gamma, \preceq)$. Furthermore, by properties 1 and 2, this poset is a distributive lattice, with join ($\vee$) and meet ($\wedge$) given by componentwise maximum and minimum. Let us now consider the poset $(\Gamma^k, \preceq^k)$ of k-tuples of feasible solutions, with product order $\preceq^k$. We say that a k-tuple $C = (X_1, X_2, \ldots, X_k)$ in Γ^k is in *left-right order* if $X_i \preceq X_j$ for all $i < j$. That is, the feasible solutions in C are arranged in non-decreasing order according to relation $\preceq$. Let $\Gamma^k_{\text{lr}} \subseteq \Gamma^k$ denote the subset of left-right ordered k-tuples.

Part 1: Distributivity. We now establish the first of the four lemmas.

▶ **Lemma 4** (Distributivity Lemma). *The poset $L^* = (\Gamma^k_{\text{lr}}, \preceq^k)$ is a distributive lattice.*

Proof. By property 2 of Theorem 1, $L = (\Gamma, \preceq)$ is a distributive lattice. Now, let $L^k = (\Gamma^k, \preceq^k)$ be the k-th power of L. We know that the product of distributive lattices is distributive; hence L^k is a distributive lattice. Moreover, since $\Gamma^k_{\text{lr}} \subseteq \Gamma^k$, the poset L^* is a sublattice of L^k. As any sublattice of a distributive lattice is itself distributive, the lemma follows. ◀

Part 2: Cost equivalence. Following the proof of the Distributivity Lemma, we now establish an equivalence between the costs of maximum diversity solutions in the sets Γ^k and Γ^k_{lr}. (Note that this is the same as establishing the equivalence between the sets Γ_k and Γ^k_{lr}, since a k-multiset over Γ has the same diversity as each of its up to $n!$ permutations – each a k-tuple – in Γ^k.) For this, we use the notion of element multiplicity. Let $C \in \Gamma^k$ be a k-tuple of solutions. The multiplicity $\mu_e(C)$ of an element $e \in E$, with respect to C, is the number of feasible solutions in C that contain e. Since a feasible solution contains no repeated elements, $\mu_e(C)$ is also the number of times e appears in the multiset sum of the solutions in C. An immediate consequence of property 1 of Theorem 1 is the following.

▶ **Observation 5.** *For any two $X, Y \in L$, we have $X \uplus Y = (X \wedge Y) \uplus (X \vee Y)$.*

Proof. Let $X = (x_1, \ldots, x_r)$ and $Y = (y_1, \ldots, y_r)$, with $x_i, y_i \in E_i$ and $i \in [r]$, where E_i is the i-th chain in the chain decomposition of E. By property 1, we know that the meet and join of two elements L are given by componentwise minimum and maximum. That is,

$$X \wedge Y = (\min(x_1, y_1), \ldots, \min(x_r, y_r)), \quad \text{and} \quad X \vee Y = (\max(x_1, y_1), \ldots, \max(x_r, y_r)).$$

Hence, if $x_i = y_i$, the element x_i appears twice in the multiset sum $X \uplus Y$ and twice in the sum $(X \wedge Y) \uplus (X \vee Y)$. If $x_i \neq y_i$, then x_i appears in either the join or the meet of X and Y, and similarly for element y_i. Finally, if an element $e \in E$ is not in $X \cup Y$, then it is neither the minimum nor maximum of any entry and therefore cannot appear in $(X \wedge Y) \cup (X \vee Y)$. Since this holds for each $i \in [k]$, the observation is proven. ◀

Observation 5 implies that the join and meet operations of the lattice L of feasible solutions preserve element multiplicities. Consequently, any k-tuple in Γ^k can be reordered into a left-right ordered form while preserving element multiplicities, as stated in the following claim (see the proof in the full version).

▷ Claim 6. For every k-tuple $C \in \Gamma^k$ there exists a left-right ordered k-tuple $\hat{C} \in L^k_{\text{lr}}$ such that $\mu_e(C) = \mu_e(\hat{C})$ for all $e \in E$.

Now, consider the pairwise-sum diversity measure introduced in Section 1. We can rewrite it directly in terms of the multiplicity as

$$d_{\text{sum}}(C) = \sum_{e \in E} \mu_e(C)(k - \mu_e(C)) = k^2 \cdot r - \sum_{e \in E} \mu_e(C)^2, \tag{1}$$

which results from observing that $\sum_{e \in E} \mu_e(C) = k \cdot r$, for any $C \in \Gamma^k$, with r the cardinality of each solution in C. This formulation highlights that maximizing d_{sum} depends only on the distribution of elements across feasible solutions, rather than their specific ordering within a tuple C. Since the terms outside the latter summation in Eq. (1) are constant, the following lemma is an immediate consequence of Claim 6.

▶ **Lemma 7** (Cost-Equivalence Lemma)**.** *For any $C \in \Gamma^k$ there exists $\hat{C} \in \Gamma^k_{\text{lr}}$ such that $d_{\text{sum}}(\hat{C}) = d_{\text{sum}}(C)$.*

Lemma 7 tells us that in order to solve Max-Sum k-Diverse Solutions, we do not need to optimize over the set of k-element multisets of Γ. Instead, we can optimize over the set Γ^k_{lr} of k-tuples that are in left-right order. Moreover, it follows from Eq. (1) that maximizing d_{sum} is equivalent to minimizing

$$\hat{d}_{\text{sum}}(C) = \sum_{e \in E} \mu_e(C)^2. \tag{2}$$

Hence, solving Max-Sum k-Diverse Solutions reduces to minimizing $\hat{d}_{\text{sum}}(C)$ in the lattice L^*. All we have left to do to complete the reduction to SFM is show that $\hat{d}_{\text{sum}}(C)$ is submodular in the lattice L^*.

Part 3: Submodularity. We begin with two claims regarding the multiplicity function $\mu_e(C)$ on L^*. These claims rely crucially on property 1 of Theorem 1. Their proofs can be found in the full version of the paper. We use $E(C)$ to denote the set of elements $\bigcup_{X \in C} X$ for a tuple $C \in \Gamma^k$.

▷ Claim 8. The multiplicity function $\mu_e : \Gamma^k_{\text{lr}} \to \mathbb{N}$ is modular on L^*.

▷ Claim 9. For any two $C_1, C_2 \in L^*$ and $e \in E(C_1) \cup E(C_2)$, it holds that $\max(\mu_e(C_1 \vee C_2), \mu_e(C_1 \wedge C_2)) \leq \max(\mu_e(C_1), \mu_e(C_2))$.

We observe that Claim 8 holds in the lattice L^k, not just in the sublattice L^* of left-right ordered k-tuples. In contrast, Claim 9 is specific to the sublattice L^*. The following proposition is an immediate consequence of these two claims and the convexity of the square function.

▶ **Proposition 10.** *For any two $C_1, C_2 \in L^*$ and any $e \in E$, we have $\mu_e(C_1 \vee C_2)^2 + \mu_e(C_1 \wedge C_2)^2 \leq \mu_e(C_1)^2 + \mu_e(C_2)^2$.*

Proposition 10 states that, for each element $e \in E$, the square of the multiplicity function μ_e is submodular in the lattice L^*. Then, taking the sum of $\mu_e(C)^2$ over all elements $e \in E$ is also a submodular function; that is

$$\sum_{e \in E} \mu_e(C_1 \vee C_2)^2 + \sum_{e \in E} \mu_e(C_1 \wedge C_2)^2 \leq \sum_{e \in E} \mu_e(C_1)^2 + \sum_{e \in E} \mu_e(C_2)^2.$$

Each sum in this 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, we obtain the following.

▶ **Lemma 11** (Submodularity Lemma). *The function $\hat{d}_{\text{sum}} : \Gamma^k_{\text{lr}} \to \mathbb{N}$ is submodular in L^*.*

Part 4: Compactness. While Lemmas 4, 7, and 11 already demonstrate the reduction of Max-Sum k-Diverse Solutions to SFM, this reduction alone does not guarantee an efficient algorithm. To complete the proof of Theorem 1, it remains to show that a compact representation of the left-right ordered lattice L^* exists and can be constructed efficiently. By Birkhoff's representation theorem, we need only specify the set of join-irreducibles of L^* to obtain a compact representation in $O(|J(L^*)|^2)$ time. This is done in the following lemma, whose proof can be found in the full version of this work.

▶ **Lemma 12** (Compactness Lemma). *The set of join-irreducibles of L^* is of size $O(kn)$ and is given by*

$$J(L^*) = \textstyle\bigcup_{i=1}^{k} J_i, \text{ where } J_i := \{(\underbrace{0_L, \ldots, 0_L}_{i-1 \text{ times}}, \underbrace{p, \ldots, p}_{k-i+1 \text{ times}}) : p \in J(L)\}.$$

With Lemma 12, a compact representation of L^* can be constructed in polynomial time. It is also clear that, given a k-tuple, the function $\hat{d}_{\text{sum}}$ can be evaluated efficiently. Then, by Theorem 3 and Lemmas 4, 7, 11, and 12, the proof of Theorem 1 is complete.

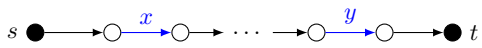


Figure 2 Illustration of the order relation $\preceq_i$ over the edges of an s-t path $p_i \in \mathcal{P}$.

4 Applications of the Framework

We now present examples of combinatorial problems whose feasible solution sets meet each of the conditions outlined in Theorem 1, allowing for the generation of maximally diverse solutions within our framework. Specifically, we discuss minimum s-t cuts (Section 4.1), stable matchings (Section 4.2), and market-clearing price vectors (Section 4.3).

4.1 Minimum s-t cuts

In the Minimum s-t Cut problem we are given a directed graph $G = (V, E)$, along with two special vertices $s, t \in V$, and are tasked with finding 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*. Here, we consider the problem of finding diverse minimum s-t cuts, formally defined below.

Max-Sum k-Diverse Minimum s-t Cuts. *Given are a directed graph $G = (V, E)$ and vertices $s, t \in V$. Let $\Gamma \subseteq 2^E$ be the set of minimum s-t cuts in G, and let Γ_k be the set of k-multisets over Γ. Find $C \in \Gamma_k$ such that $d_{\text{sum}}(C) = \max_{S \in \Gamma_k} d_{\text{sum}}(S)$.*

Using our framework, we reproduce the findings of De Berg et al. [7] for Max-Sum k-Diverse Minimum s-t Cuts by showing that the set Γ of minimum s-t cuts satisfies properties 1-3 of Theorem 1. We prove these statements in order.

▶ **Lemma 13** (Property 1). *There is a chain decomposition of the edge set E into r disjoint chains, such that each minimum s-t cut $X \in \Gamma$ contains exactly one element from each chain.*

Proof. We construct the r chains as follows. Let $\mathcal{P}$ be an (arbitrary) set of edge-disjoint s-t paths in G with maximum cardinality r. Define $E(p_i)$ as the set of edges traversed by the path $p_i \in \mathcal{P}$. For each path $p_i \in \mathcal{P}$, consider the order relation $\preceq_i$ defined as follows: for any $x, y \in E(p_i)$, we say $x \preceq_i y$ if and only if path p_i meets edge x before edge y, or if x and y are the same edge. Since every pair of edges within a path p_i is comparable under this relation, each poset $(E(p_i), \preceq_i)$, for $i \in [r]$, forms a chain. Moreover, these chains are disjoint by the definition of the set $\mathcal{P}$. By Menger's theorem, the size of a minimum s-t cut in G equals the maximum number of edge-disjoint s-t paths, which is r. Consequently, any minimum s-t cut $X \subseteq E$ must include exactly one edge from each chain $(E(p_i), \preceq_i)$, $i \in [r]$. Otherwise, if X contained fewer than r edges, it would not be a valid s-t cut, and if it contained more, it would not be of minimum size. Hence, $\Gamma \subseteq E(p_1) \times \cdots \times E(p_r)$.

Consider now the edges in $E' = E \setminus \bigcup_{1 \leq i \leq r} E(p_i)$. We call these edges *residual edges*. Observe that these edges can never be part of a minimum s-t cut. This follows because such a cut must contain exactly one edge from each chain in $\mathcal{P}$, and cutting any additional edge from E' would only increase the cut size, violating minimality. Hence, we simply distribute the residual edges arbitrarily over the r chains. This does not change the fact that the chains are disjoint, or that the set of minimum s-t cuts is a subset of the cartesian product of the augmented chains. This completes the proof. ◀

By Lemma 13, the set $\Gamma \subseteq E(p_1) \times \cdots \times E(p_r)$ of minimum s-t cuts with componentwise order – defined by: $(x_1, \ldots, x_r) \preceq (y_1, \ldots, y_r)$ for $(x_1, \ldots, x_r), (y_1, \ldots, y_r) \in \Gamma$ iff $x_i \preceq_i y_i$ for all $i \in [r]$ – forms a poset $L = (\Gamma, \preceq)$. It is well known that this poset defines a distributive

lattice [11, 31, 22]. Specifically, proving that Γ is closed under the joins and meets induced by $\preceq$ suffices to establish this property (see e.g., [7, Claim A.1]). Thus, property 2 of Theorem 1 follows directly.

▶ **Lemma 14** (Property 2). *The set Γ of minimum s-t cuts with componentwise order $\preceq$ defines a distributive lattice L.*

Next, we note that a compact representation of the lattice of minimum s-t cuts can be constructed in polynomial time. This result is well known from the work of Picard and Queyranne [35], who gave an algorithm to build such a representation using a residual graph. Specifically, the resulting graph has vertex set $J(L) \cup 0_L$ and total size at most $|V|$, and can therefore be constructed in $O(|V|^2)$ time.

▶ **Lemma 15** (Property 3). *A compact representation of the lattice s-t mincuts can be constructed in polynomial time.*

Then, by Theorem 1 and Lemmas 13-15, we obtain a polynomial time algorithm for Max-Sum k-Diverse Minimum s-t Cuts via submodular function minimization.

▶ **Theorem 16.** *Max-Sum k-Diverse Minimum s-t Cuts is poly-time solvable.*

▶ Remark 17. Since the edge version of Menger's theorem is known to hold for multigraphs, our results for unweighted directed graphs extend naturally to weighted graphs by replacing each edge of weight w with w parallel edges.

▶ Remark 18. Similar results to those presented in Lemmas 13-15 can be established for *minimum s-t vertex cuts*. Since a vertex-connectivity version of Menger's theorem also exists, the arguments in Lemma 13 remain valid when replacing E with V. Moreover, the poset of minimum s-t vertex cuts, ordered componentwise, forms a distributive lattice, which can be demonstrated analogously to Lemma 14. Finally, a compact representation of this lattice can be computed efficiently, as shown by Bonsma [5, Sec. 6], or via the constructive version of Birkhoff's theorem for computing a *slice*, as described by Garg [33] (see also [15, Ch. 10]).

4.2 Stable Matchings

In the Stable Matching problem, we are given a complete bipartite graph $K_{n,n} = (A \cup B, E)$ along with a linear ordering $\preceq_a$ over B for each vertex $a \in A$, and similarly a linear ordering $\preceq_b$ over A for each vertex $b \in B$. For a vertex $a \in A$ (resp. $b \in B$), the poset $L_a = (B, \preceq_v)$ (resp. $L_b = (A, \preceq_v)$) is referred to as its *preference list*. The task is to find a perfect matching M in $K_{n,n}$ such that no two vertices $a \in A$ and $b \in B$ prefer each other over their matched partners. Such a set of edges is called a *stable matching*. We now consider the problem of finding diverse stable matchings.

Max-Sum k-Diverse Stable Matching. *Given are a complete bipartite graph $K_{n,n} = (A \cup B, E)$, along with preference lists L_a and L_b for each $a \in A$ and $b \in B$. Let $\Gamma \subseteq 2^E$ be the set of stable matchings in G, and let Γ_k denote the set of k-multisets over Γ. Find $C \in \Gamma_k$ such that $d_{\text{sum}}(C) = \max_{S \in \Gamma_k} d_{\text{sum}}(S)$.*

We show that Max-Sum k-Diverse Stable Matching can be solved in polynomial time by proving that the set Γ of stable matchings satisfies properties 1-3 of Theorem 1.

▶ **Lemma 19** (Property 1). *There is a chain decomposition of the edge set E into r disjoint chains, such that each stable matching $X \in \Gamma$ contains exactly one element from each chain.*

Proof. Let $r = n$. Note that the posets L_a and L_b are chains. We claim that the chains L_a, $a \in A$ define a disjoint chain decomposition of the ground set E.[1] First, we argue for disjointness. Let $E(a) = \{(a, b) \mid b \in L_a\}$ denote the set of edges defined by the preference list L_a of an arbitrary vertex $a \in A$. Since $K_{n,n}$ is bipartite, there are no edges between the vertices of A. This implies that $E(a_1) \cap E(a_2) = \emptyset$ for all distinct $a_1, a_2 \in A$. Moreover, $E = \bigcup_{a \in A} E(a)$. Hence, the chains L_a, $a \in A$ define a disjoint chain decomposition of E.

Now, we argue that a stable matching must contain exactly one element from each chain L_a. This follows immediately from the definition of perfect matching, which requires every vertex in A to be matched to exactly one vertex in B. Consequently, each stable matching selects precisely one edge from $E(a)$ for each $a \in A$. This completes the proof. ◀

To establish property 2, we use the following well-known result from the stable matchings literature [28, 3].

▷ Claim 20 ([28, Thm. 7 & Cor. 1]). Given any two stable matchings $X = ((a_1, b_1), \ldots, (a_n, b_n))$ and $Y = ((a_1, b'_1), \ldots, (a_n, b'_n))$, then

$$X \vee Y = ((a_1, \max_{\preceq_{a_1}}(b_1, b'_1)), \ldots, (a_n, \max_{\preceq_{a_n}}(b_n, b'_n))) \quad \text{and}$$
$$X \wedge Y = ((a_1, \min_{\preceq_{a_1}}(b_1, b'_1)), \ldots, (a_n, \min_{\preceq_{a_n}}(b_n, b'_n)))$$

are also stable matchings.

By standard results in lattice theory (see e.g., [19]), the cartesian product $E_{\text{prod}} = E(a_1) \times \cdots \times E(a_n)$, with componentwise order $\preceq$, forms a distributive lattice $(E_{\text{prod}}, \preceq)$. Then, by Lemma 19 and Claim 20, the poset $L = (\Gamma, \preceq)$ is a sublattice of E_{prod}, which implies that L is also distributive.

▶ **Lemma 21** (Property 2). *The set Γ of stable matchings with componentwise order $\preceq$ defines a distributive lattice L.*

It only remains to verify that property 3 of Theorem 1 is satisfied by the set Γ of stable matchings. This property follows directly, since the so-called *poset of rotations* introduced by Gusfield [21] provides the required structure (see also, e.g., [14, Sec 2.3]).

▶ **Lemma 22** (Property 3 [21, Lemma 3.3.2]). *A compact representation of the lattice L of stable matchings can be constructed in $O(|V|^2)$ time.*

Then, by Theorem 1 and Lemmas 19-22, the following theorem holds.

▶ **Theorem 23.** *Max-Sum k-Diverse Stable Matching is poly-time solvable.*

4.3 Market Clearing Price Vectors

As a final example, we consider the problem of finding an integer *market-clearing price* in a *matching market* (see e.g., [10, Ch. 10] for details). Such a market consists of a set I of n items and a set U of n bidders, where each item can be assigned to at most one bidder, and each bidder wants to buy at most one item. Each bidder $b \in U$ assigns a valuation $v_{b,i}$ to every item $i \in I$, where $v_{b,i}$ is an integer between 0 and T. We assume that $T = poly(n)$.

[1] Note that we may also choose the chains L_b, $b \in B$ and get similar results.

The Market Clearing Price problem asks for a price vector P assigning a price $P[i] \in [0, T]$ to each item i, such that the bipartite graph $(I, U, E(P))$ defined by

$$(j, b) \in E(P) \iff \forall i \in I : (v_{b,j} - P[j]) \geq (v_{b,i} - P[i])$$

has a perfect matching. (Informally, an edge connects item j to bidder b if j gives b their highest payoff – valuation minus price.) A price vector P that yields such a graph is called a *market-clearing price* vector.

We now turn to the problem of finding diverse market-clearing price vectors.

Max-Sum k-Diverse Market Clearing Price. *Given is a matching market $M = (I, U, v)$ where I is a set of n items, U is a set of n bidders, and $v : I \times U \to [0, T]$ is a valuation function. Let $\Gamma \subseteq [T]^n$ be the set of market-clearing price vectors in M, and let Γ_k be the set of k-multisets over Γ. Find $C \in \Gamma_k$ such that $d_{\text{sum}}(C) = \max_{S \in \Gamma_k} d_{\text{sum}}(S)$.*

At this point, the reader may wonder whether d_{sum} is a good choice for vectors; we return to this question later. For now, we show that Max-Sum k-Diverse Market Clearing Price can be solved in polynomial time by verifying that Γ satisfies properties 1–3 of Theorem 1. The first property follows immediately from the definition of a price vector.

▶ **Lemma 24** (Property 1). *There is a chain decomposition of the ground set E into r disjoint chains, such that each market-clearing price vector contains exactly one element from each chain.*

Proof. By definition, a market-clearing price vector is an element of the power set $E_{\text{prod}} = [0, T]^n$. Hence, the chain decomposition of E_{prod} consists of n copies of the integer interval $E = [0, T]$, with elements from different chains deemed incomparable. The ordering $\preceq_i$ of each chain E_i is the natural ordering. ◀

It is clear that the power set E_{prod} with product order $\preceq^n$ forms a distributive lattice, where join ($\vee$) and meet ($\wedge$) are determined by componentwise maximum and minimum. Shapley et al. [37] (see also [39]) establish that the poset $L = (\Gamma, \preceq^n)$ is closed under these join and meet operations, hence L is a sublattice of E_{prod}, and thus, L is distributive.

▶ **Lemma 25** (Property 2 [37]). *The poset $(\Gamma, \preceq^n)$ is a distributive lattice, with the meet and join defined appropriately.*

As for property 3, Garg [16, 17] recently established that the set $J(L)$ of join-irreducibles of the lattice L of market-clearing price vectors can be determined efficiently (in polynomial time) via an algorithm for detecting so-called *lattice-linear predicates* (see also [18] and [15, Ch. 10]). We thus get the following result.

▶ **Lemma 26** (Property 3 [17]). *A compact representation of the lattice L of market-clearing price vectors can be constructed in polynomial time.*

Then, by Theorem 1 and Lemmas 24, 25, and 26, the following theorem holds.

▶ **Theorem 27.** *Max-Sum k-Diverse Market-Clearing Price is poly-time solvable.*

Let us briefly reflect on the choice of diversity measure. The pairwise-sum measure d_{sum} captures whether elements differ, but not by how much; an important aspect for price vectors, where the magnitude of values matters more than their identity (unlike in cuts or matchings). In the next section, we extend our framework to support alternative measures, such as the *absolute-difference* measure, which better captures diversity in price vectors.

5 Other Diversity Measures

The proof of Theorem 1 relies on four lemmas, with the diversity measure playing a role in only two of them: the Cost-Equivalence (Lemma 7) and Submodularity (Lemma 11) lemmas. For simplicity, we have presented our main result in terms of the d_{sum} diversity measure. However, the framework is not limited to this specific measure. Just as it applies to problems whose solution sets satisfy the properties of Theorem 1, it also extends to other diversity measures, provided they satisfy both the Cost-Equivalence and Submodularity lemmas.

Here, we mention two such diversity measures: the *coverage* diversity d_{cov}, and the L_1- or *absolute-difference* diversity d_{abs}. Let E be a finite set with n elements, and let $\Gamma \subseteq 2^E$ be a set of feasible solutions. Given a k-tuple of feasible solutions $(X_1, \ldots, X_k) \in \Gamma^k$, these measures are defined as follows:

$$d_{\text{cov}}(X_1, X_2, \ldots, X_k) = \bigcup_{1 \le i \le k} |X_i|, \quad \text{and} \tag{3}$$

$$d_{\text{abs}}(X_1, X_2, \ldots, X_k) = \sum_{1 \le i < j \le k} f(X_i, Y_j), \tag{4}$$

where $f(X, Y) = \sum_i^r \|x_i - y_i\|$ for any two $X = (x_1, \ldots, x_r), Y = (y_1, \ldots, y_r) \in \Gamma$.

The coverage diversity measures the number of distinct elements appearing across solutions, while the absolute-difference diversity quantifies diversity by summing coordinate-wise differences between solutions. Notice that the latter applies only to solutions representable as r-tuples, since f requires component-wise comparisons, and it assumes a notion of difference between elements in E (e.g., r-dimensional integer vectors in $[-M, M]^r$, with $M \in \mathbb{N}$).

Each of these two measures gives rise to a corresponding optimization problem, defined in the same way as Max-Sum Diverse Solutions but with d_{sum} replaced by d_{cov} or d_{abs}. We refer to these as Max-Cov k-Diverse Solutions and Max-Abs k-Diverse Solutions, respectively. Our main result for these problems is the following:

▶ **Theorem 28.** *Max-Cov k-Diverse Solutions and Max-Abs k-Diverse Solutions can be solved in polynomial time if the set of feasible solutions Γ satisfies the three properties of Theorem 1.*

We establish this result by showing that both d_{cov} and d_{abs} satisfy the Cost-Equivalence and Submodularity lemmas from Section 3. Due to space constraints, the proofs for d_{cov} are deferred to the full version (see also [7, Thm. 14]). Here, we prove these lemmas for d_{abs}.

Proof of Cost-Equivalence. The following lemma is an immediate consequence of Claim 6, which states that any k-tuple in Γ^k can be reordered into a left-right ordered form while preserving element multiplicities.

▶ **Lemma 29** (Cost-Equivalence Lemma)**.** *Let $C \in \Gamma^k$ such that $d_{abs}(C) = \max_{S \in \Gamma^k} d_{\text{abs}}(S)$. Then there exists $\hat{C} \in \Gamma^k_{\text{lr}}$ such that $d_{\text{abs}}(\hat{C}) = d_{\text{abs}}(C)$.*

Proof. Let $C \in \Gamma^k$ be an arbitrary k-tuple of solutions, and let $\hat{C} \in \Gamma^k_{\text{lr}}$ be its reordering into left-right order by the algorithm of Claim 6. For a feasible solution $X \in \Gamma$, let $X(\ell)$ denote its ℓ-th component.

Consider the k-tuples $C(\ell) = (X_1(\ell), \ldots, X_k(\ell))$ and $\hat{C}(\ell) = (\hat{X}_1(\ell), \ldots, \hat{X}_k(\ell))$, where $X_i \in C$, $\hat{X}_i \in \hat{C}$ for all $i \in [k]$. These represent the ℓ-th component of each solution in C and $\hat{C}$, respectively. Now, define the function $f_\ell : E_\ell^k \to \mathbb{R}$ as:

$$f_\ell(x_1, \ldots, x_k) = \sum_{1 \le i < j \le k} \|x_i - x_j\| .$$

By Claim 6, the multiplicity of each element in $C(\ell)$ is preserved in $\hat{C}(\ell)$, implying that $f_\ell(C(\ell)) = f_\ell(\hat{C}(\ell))$. Since the absolute-difference diversity measure decomposes as:

$$d_{\text{abs}}(X_1, \ldots, X_k) = \sum_{\ell=1}^{r} f_\ell(X_1(\ell), \ldots, X_k(\ell)),$$

it follows that $d_{\text{abs}}(\hat{C}) = d_{\text{abs}}(C)$. In particular, this holds for tuples that achieve maximum diversity. ◀

Proof of Submodularity. In this case, the Submodularity Lemma actually becomes a Modularity Lemma. First, consider the function $f'_\ell : \Gamma^2_{\text{lr}} \to \mathbb{R}$ defined by $f'_\ell(X_1, X_2) = \|X_1(\ell) - X_2(\ell)\|$, where $X_1 \preceq X_2$, $\ell \in [r]$, and $X(\ell)$ denotes the ℓ-th component of a solution $X \in \Gamma$. We can rewrite the absolute difference diversity measure as:

$$d_{\text{abs}}(X_1, \ldots, X_k) = \sum_{\ell}^{r} \sum_{1 \le i < j \le k} f'_\ell(X_i, X_j).$$

If we can establish that $f'_\ell(\cdot)$ is modular in L, then, because the sum of modular functions is modular, d_{abs} would also be modular. We prove this in the following lemma.

▶ **Lemma 30** (Modularity Lemma). *The function $\hat{d}_{\text{abs}} : \Gamma^k_{\text{lr}} \to \mathbb{N}$ is modular in L^*.*

Proof. We prove that, for any two $C_1, C_2 \in \Gamma^2_{\text{lr}}$, the function $f'_\ell(\cdot)$ is modular in the lattice $(\Gamma^2_{\text{lr}}, \preceq^2)$, where $\preceq$ is the componentwise order of the poset L of feasible solutions. Since the sum of modular functions is modular, and d_{abs} can be written as a sum of functions $f'_\ell(\cdot)$ over all $\ell \in [r]$, the modularity of d_{abs} follows.

Let $C_1 = (X_1, X_2)$ and $C_2 = (Y_1, Y_2)$. By definition, the join ($\vee$) and meet ($\wedge$) of C_1 and C_2 are given by componentwise maximum and minimum. Then, for each $\ell \in [r]$,

$$f'_\ell(C_1 \wedge C_2) = \|\min(X_1(\ell), Y_1(\ell)) - \min(X_2(\ell), Y_2(\ell))\| \quad \text{and}$$
$$f'_\ell(C_1 \vee C_2) = \|\max(X_1(\ell), Y_1(\ell)) - \max(X_2(\ell), Y_2(\ell))\|.$$

Consider an arbitrary $\ell \in [r]$. Because C_1 and C_2 are each in left-right order, we have: $X_1(\ell) \preceq X_2(\ell)$ and $Y_1(\ell) \preceq Y_2(\ell)$. Consider then the intervals $I_X = [X_1(\ell), X_2(\ell)]$ and $I_Y = [Y_1(\ell), Y_2(\ell)]$. Without loss of generality, assume that $X_2(\ell) \preceq Y_2(\ell)$. There are three possibilities for the interaction of I_X and I_Y: (i) the intervals are disjoint (i.e., $I_X \cap I_Y = \emptyset$), they overlap (i.e., $I_X \cap I_Y \neq \emptyset$), or (iii) one is contained in the other (i.e., $I_X \subset I_Y$). We now compare the sums $f'_\ell(C_1 \wedge C_2) + f'_\ell(C_1 \vee C_2)$ and $f'_\ell(C_1) + f'_\ell(C_2)$ in each of these cases.

In cases (i) and (ii), we have that $X_1(\ell) \preceq Y_1(\ell)$ and $X_2(\ell) \preceq Y_2(\ell)$. Hence,

$$f'_\ell(C_1 \wedge C_2) + f'_\ell(C_1 \vee C_2) = \|X_1(\ell) - X_2(\ell)\| + \|Y_1(\ell) - Y_2(\ell)\| = f'_\ell(C_1) + f'_\ell(C_2),$$

and thus, modularity is satisfied.

In case (iii), we have $Y_1(\ell) \preceq X_1(\ell)$ and $X_2(\ell) \preceq Y_2(\ell)$. Then:

$$\begin{aligned} f'_\ell(C_1 \wedge C_2) + f'_\ell(C_1 \vee C_2) &= \|Y_1(\ell) - X_2(\ell)\| + \|X_1(\ell) - Y_2(\ell)\| \\ &= (X_2(\ell) - Y_1(\ell)) + (Y_2(\ell) - X_1(\ell)) \\ &= (X_2(\ell) - X_1(\ell)) + (Y_2(\ell) - Y_1(\ell)) \\ &= f'_\ell(C_1) + f'_\ell(C_2), \end{aligned}$$

which again satisfies modularity.

Therefore, the function $f'_\ell(\cdot)$ is modular in $(\Gamma^2_{\text{lr}}, \preceq^2)$, and the lemma is proved. ◀

By replacing the Cost-Equivalence and Submodularity lemmas of Section 3 with Lemmas 29 and 30 above, the proof of (the second half of) Theorem 28 is complete.

6 A Simple Framework for Disjoint Solutions

Finally, we consider the special case of diversity where solutions are required to be pairwise disjoint. Specifically, we consider the problem Max-Disjoint Solutions, defined below, and outline an algorithm for solving it that bypasses the need for submodular function minimization, provided that properties 1 and 2 of Theorem 1 are satisfied.

Max-Disjoint Solutions. *Given a finite set E of size n, an implicitly defined family Γ of subsets of E, referred to as feasible solutions, and a membership oracle $\mathcal{O}_\Gamma$ for Γ, find a set $C \subseteq \Gamma$ such that $X \cap Y = \emptyset$ for all $X, Y \in C$, and $|C|$ is as large as possible.*

We assume that the set Γ of feasible solutions satisfies properties 1 and 2 of Theorem 1. That is, there is a poset $P = (E, \leq)$ that is the disjoint union of r chains $(E_i, \preceq_i)$, $i \in [r]$, and the set $\Gamma \subseteq E_1 \times \cdots \times E_r$, with componentwise order $\preceq$, forms a distributive lattice.

The idea behind the algorithm is simple: start by finding the bottom element of the lattice of feasible solutions (using the $\mathcal{O}_{\min}$ oracle below), remove it along with any other solutions that overlap with it (enabled by the $\mathcal{O}_{\text{ds}}$ oracle below), and then repeat this process on the remaining sublattice until no feasible solutions remain. Of course, we want to avoid working on the lattice directly, as it can be of exponential size. Instead, we assume that the algorithm has access to the following oracles, or subroutines:

- Minimal/Maximal Solution Oracles ($\mathcal{O}_{\min}$ and $\mathcal{O}_{\max}$): On input $\langle P, \mathcal{O}_\Gamma \rangle$, the *minimal solution oracle* ($\mathcal{O}_{\min}$) returns the bottom element of the distributive lattice $(\Gamma, \preceq)$, while the *maximal solution oracle* ($\mathcal{O}_{\max}$) returns its top element; i.e.,

$$\mathcal{O}_{\min}(P, \mathcal{O}_\Gamma) = \bigwedge_{X \in \Gamma} X, \quad \text{and} \quad \mathcal{O}_{\max}(P, \mathcal{O}_\Gamma) = \bigvee_{X \in \Gamma} X.$$

- Disjoint Successors Oracle ($\mathcal{O}_{\text{ds}}$): For a feasible solution $X \in \Gamma$, the subset $\Gamma(X) \subset \Gamma$ of *disjoint successors of X* consists of all feasible solutions that are both disjoint from X and successors of X with respect to the order $\preceq$; that is, $\Gamma(X) = \{Y \mid Y \in \Gamma, X \cap Y = \emptyset, X \preceq Y\}$. Given an input $\langle X, P, \mathcal{O}_\Gamma \rangle$, this oracle returns the subposet of P induced by the subset of elements of E that appear in the disjoint succesors of X; i.e.,

$$\mathcal{O}_{\text{ds}}(X, E, \mathcal{O}_\Gamma) = P\left[\bigcup \Gamma(X)\right].$$

In this general framework, we achieve the following result.

▶ **Theorem 31.** *Max-Disjoint Solutions can be solved in $O(n)$ oracle calls.*

In other words, if subroutines $\mathcal{O}_{\min}$, $\mathcal{O}_{\max}$, and $\mathcal{O}_{\text{ds}}$ could be computed in polynomial time, we could solve Max-Disjoint Solutions in polynomial time as well. Note that these subroutines are problem-specific and must be designed and implemented based on the particular problem defined by P and Γ. This framework has been applied implicitly in prior work on disjoint minimum s-t cuts [7] and stable matchings [13], where the oracles above run in near-linear time. Our algorithm extends these ideas to a more general setting – namely, to problems satisfying properties 1 and 2 of Theorem 1.

Next, we give a formal description of the algorithm and prove its correctness and complexity, completing the proof of Theorem 31.

Preliminaries. Before we formally describe the algorithm, we require some results and notation. Throughout, let L denote the distributive lattice $(\Gamma, \preceq)$. We use X_z and X_o to denote the top and bottom elements of a lattice L, respectively, which are the two elements

that satisfy $X_o \preceq X \preceq X_z$ for all $X \in \Gamma$. For a feasible solution $X \in \Gamma$, we use $X(\ell)$ to denote the element in the ℓ-th component of X. Note that $X(\ell) \in E_\ell$. The following observation is a necessary condition for the existence of disjoint solutions in Γ.

▶ **Observation 32.** *Let $e \in E$ be an element of both X_o and X_z. Then e must be present in every feasible solution in Γ.*

Proof. Consider an arbitrary feasible solution $X \in \Gamma$. Without loss of generality, let $e \in E_\ell$. By definition of bottom element of L, we have $X_o \preceq X$, and thus $e \preceq_\ell X(\ell)$. On the other hand, by definition of top element of L, we have $X \preceq X_z$, which implies $X(\ell) \preceq_\ell e$. By antisymmetry of the partial order $\preceq_\ell$, it follows that $X(\ell) = e$. Hence, $e \in X$, proving the fact. ◀

We also make the following observation about the set of disjoint successors of a feasible solution. With a slight abuse of notation, we use $\preceq$ to denote to the componentwise ordering arising from P and any induced suposet.

▶ **Observation 33.** *For any $X \in \Gamma$, the set $\Gamma(X)$ of disjoint successors of X satisfies properties 1 and 2 of theorem 1.*

Proof. We start with the first property. Let $P(X) = [\bigcup \Gamma(X)]$ denote the subposet induced by $\bigcup \Gamma(X)$. This subposet then consists of the disjoint chains of P but restricted to the elements appearing in $\bigcup \Gamma(X)$. By definition of both $P(X)$ and $\Gamma(X)$, each solution in $\Gamma(X)$ must contain exactly one element from each chain in $P(X)$. Hence, property 1 is satisfied.

As for the second property, it is clear that $\Gamma(X) \subset \Gamma$. Moreover, because the join ($\vee$) and meet ($\wedge$) operations in L are defined as the componentwise maximum and minimum, respectively, $\Gamma(X)$ remains closed under these operations. This means that the poset $(\Gamma(X), \preceq)$ is a sublattice of L and thus, a distributive lattice. Hence, property 2 is satisfied. ◀

With these results, we are ready to describe and analyze the algorithm.

The algorithm. Given an input $\langle P, \mathcal{O}_\Gamma \rangle$, the algorithm begins by determining the bottom element X_o and the top element X_z of lattice L by querying the oracles $\mathcal{O}_{\min}$ and $\mathcal{O}_{\max}$ with the input $\langle P, \mathcal{O}_\Gamma \rangle$. If these two solutions share an element, the algorithm stops, as Observation 32 ensures that no disjoint solutions exist. Otherwise, it proceeds by querying $\mathcal{O}_{\text{ds}}(X_o, P, \mathcal{O}_\Gamma)$ to determine the subposet $P(X_o)$ induced by $\bigcup \Gamma(X_o)$.

By Observation 33, the set $\Gamma(X_o)$ satisfies properties 1 and 2 of Theorem 1, with the poset $P(X_o)$ serving as the corresponding chain decomposition. Let $L(X_o) = (\Gamma(X_o), \preceq)$ be the associated sublattice of disjoint successors of X_o. The algorithm proceeds by querying $\mathcal{O}_{\min}$ with the input $\langle P(X_o), \mathcal{O}_\Gamma \rangle$ to identify the bottom element X'_o of $L(X_o)$. Once more, if X'_o is disjoint from X_z, the algorithm queries $\mathcal{O}_{\text{ds}}(X'_o, P(X_o), \mathcal{O}_\Gamma)$ to determine the subposet $P(X'_o)$ induced by the set $\bigcup \Gamma(X'_o)$ of disjoint successors of X'_o. This process repeats as long as $\mathcal{O}_{\min}$ continues to return solutions that are disjoint from X_z. Throughout the execution, the algorithm maintains a set C that stores all solutions found that are disjoint from X_z and returns this set upon termination. The algorithm is presented below as Algorithm 1.

Correctness. The solutions in the set $C = \{X_1, X_2, \ldots, X_k\}$ returned by Algorithm 1 are clearly disjoint by construction, as the poset returned by the oracle $\mathcal{O}_{\text{ds}}$ at each step is induced by the set of disjoint successors of the solution identified in the precious step. Moreover, the set C is, in fact, a left-right ordered tuple. This follows again by construction, as each newly

Algorithm 1 Max-Disjoint Solutions.

Input: A poset P and a membership oracle $\mathcal{O}_\Gamma$ satisfying properties 1 and 2 of Theorem 1.
Output: A maximum cardinality set C of disjoint feasible solutions from Γ.

$C \leftarrow \emptyset$
$X_z \leftarrow \mathcal{O}_{\max}(P, \mathcal{O}_\Gamma)$ ▷ Top element of lattice L.
$X \leftarrow \mathcal{O}_{\min}(P, \mathcal{O}_\Gamma)$ ▷ Bottom element of lattice L.
$P(X) \leftarrow \mathcal{O}_{\text{ds}}(P, X_o, \mathcal{O}_\Gamma)$ ▷ This defines a new instance.
while $X \cap X_z = \emptyset$ **do**
    $C \leftarrow C \cup \{X\}$
    $X \leftarrow \mathcal{O}_{\min}(P(X), \mathcal{O}_\Gamma)$ ▷ New disjoint solution.
    $P(X) \leftarrow \mathcal{O}_{\text{ds}}(P, X, \mathcal{O}_\Gamma)$
$C \leftarrow C \cup \{X\}$
return C

identified solution is determined from the subset of elements that are chain-successors of elements included in previously identified solutions. Note that the notion of left-right order here is strict, meaning that $X_i \prec X_j$ for any $1 \leq i < j \leq k$.

To analyze this further, let us go back for a moment to Section 3. Note that the d_{sum} measure is maximum whenever its input consists of disjoint solutions. Consider then an arbitrary k-tuple of disjoint feasible solutions, for some $k > 0$. We know, by Claim 6, that there exists a k-tuple of disjoint feasible solutions that is in left-right order. In particular, this is true for a disjoint-solutions tuple of maximum cardinality k^*. Then, as we did in Section 3, we may restrict our arguments to the set of k^*-tuples that are in left-right order without loss of generality.

To complete the correctness of Algorithm 1, it remains to show that the tuple returned by the algorithm is of maximum cardinality k^*.

▶ **Lemma 34.** *Algorithm 1 outputs a longest tuple of disjoint feasible solutions.*

Proof. Let $C_{\text{ALG}} = (X_1, X_2, \ldots, X_k)$ be the k-tuple of disjoint feasible solutions returned by Algorithm 1. For the sake of contradiction, suppose that $C' = (Y_1, Y_2, \ldots, Y_\ell)$ is a longest left-right ordered tuple of disjoint feasible solutions with $\ell > k$.

By definition of bottom element, we know that solution $X_1 = X_o$ is a predecessor of every other feasible solution in Γ. This implies that $Y_1 \cap X_1 \neq \emptyset$; otherwise, we could append X_1 to the start of C' and obtain a longer tuple of left-right ordered disjoint solutions. Then, we have $X_1 \preceq Y_1 \prec Y_2$, and we may replace Y_1 in C' with X_1 to generate a new ℓ-tuple C_1 of disjoint solutions.

By Observation 33, and the definition of bottom element, we know that solution X_2 found by the algorithm is a predecessor of every feasible solution in $\Gamma(X_1)$; that is, X_2 is a predecessor of every feasible solution disjoint from X_1. By the same argument as before, $X_2 \cap Y_2 \neq \emptyset$. We then have $X_2 \preceq Y_2 \prec Y_3$, and we may replace Y_2 in C_1 with X_2 to generate a new ℓ-tuple C_2 of disjoint solutions.

By repeating this procedure k times, we end up with the ℓ-tuple $C_k = (X_1, X_2, \ldots, X_k, Y_{k+1}, \ldots, Y_\ell)$ of left-right ordered disjoint solutions. Then, there exists a feasible solution Y_{k+1} that is a strict successor of X_k – the last element of tuple C_{ALG}. But this implies that X_k is disjoint with the top element X_z of L, which we know to be false by construction of C_{ALG}. Thus, we get the necessary contradiction. ◀

Time complexity. The oracles $\mathcal{O}_{\min}$ and $\mathcal{O}_{\text{ds}}$ are called k^* times, and k^* is upper bounded by the length of the shortest chain in P, which, in the worst case, has length $O(n)$. This completes the proof of Theorem 31.

7 Concluding Remarks

We showed that Max-Cov k-Diverse Solutions can be solved in polynomial time by reducing it to submodular function minimization on a distributive lattice, provided the feasible solution set satisfies three structural properties. This gives a general framework for designing polynomial-time algorithms for diverse variants of combinatorial problems. We applied it to Minimum s-t Cut, Stable Matching, and Market Clearing Price, and showed it extends beyond the pairwise-sum measure to the coverage and absolute-difference measures. We also showed that in the special case where diversity is defined by pairwise disjointness, the problem can be solved without relying on submodular function minimization.

A natural direction for future work is identifying more problems that satisfy the structural properties of Theorem 1, and to characterize which diversity measures are compatible with the framework. It remains open whether the reliance on SFM can be avoided without losing generality.

References

1 Julien Baste, Lars Jaffke, Tomáš Masařík, Geevarghese Philip, and Günter Rote. Fpt algorithms for diverse collections of hitting sets. *Algorithms*, 12(12):254, 2019. doi:10.3390/A12120254.

2 Garrett Birkhoff. Rings of sets. *Duke Mathematical Journal*, 3(3):443–454, 1937.

3 Charles Blair. The lattice structure of the set of stable matchings with multiple partners. *Mathematics of operations research*, 13(4):619–628, 1988. doi:10.1287/MOOR.13.4.619.

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 Paul Bonsma. Most balanced minimum cuts. *Discrete Applied Mathematics*, 158(4):261–276, 2010. doi:10.1016/J.DAM.2009.09.010.

6 Brian A Davey and Hilary A Priestley. *Introduction to lattices and order*. Cambridge university press, 2002.

7 Mark de Berg, Andrés López Martínez, and Frits Spieksma. Finding diverse minimum st cuts. In *34th International Symposium on Algorithms and Computation*, 2023.

8 Mark de Berg, Andrés López Martínez, and Frits Spieksma. Finding diverse solutions in combinatorial problems with a distributive lattice structure, 2025. doi:10.48550/arXiv.2504.02369.

9 Karolina Drabik and Tomáš Masařík. Finding diverse solutions parameterized by cliquewidth. *arXiv preprint*, 2024. arXiv:2405.20931.

10 David Easley, Jon Kleinberg, et al. *Networks, crowds, and markets: Reasoning about a highly connected world*, volume 1. Cambridge university press Cambridge, 2010.

11 Fernando Escalante. Schnittverbände in graphen. In *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, volume 38, pages 199–220. Springer, 1972.

12 Fedor V Fomin, Petr A Golovach, Lars Jaffke, Geevarghese Philip, and Danil Sagunov. Diverse pairs of matchings. *Algorithmica*, 86(6):2026–2040, 2024. doi:10.1007/S00453-024-01214-7.

13 Aadityan Ganesh, HV Vishwa Prakash, Prajakta Nimbhorkar, and Geevarghese Philip. Disjoint stable matchings in linear time. In *Graph-Theoretic Concepts in Computer Science: 47th International Workshop, WG 2021, Warsaw, Poland, June 23–25, 2021, Revised Selected Papers 47*, pages 94–105. Springer, 2021. doi:10.1007/978-3-030-86838-3_7.

14 Rohith Reddy Gangam, Tung Mai, Nitya Raju, and Vijay V Vazirani. A structural and algorithmic study of stable matching lattices of "nearby" instances, with applications. In *42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.FSTTCS.2022.19.

15 Vijay K Garg. *Introduction to lattice theory with computer science applications*. John Wiley & Sons, 2015.

16 Vijay K Garg. Applying predicate detection to the constrained optimization problems. *arXiv preprint*, 2018. arXiv:1812.10431.

17 Vijay K Garg. Predicate detection to solve combinatorial optimization problems. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures*, pages 235–245, 2020. doi:10.1145/3350755.3400235.

18 Vijay K Garg and Neeraj Mittal. On slicing a distributed computation. In *Proceedings 21st International Conference on Distributed Computing Systems*, pages 322–329. IEEE, 2001. doi:10.1109/ICDSC.2001.918962.

19 George Gratzer. *Lattice theory: First concepts and distributive lattices*. Courier Corporation, 2009.

20 Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric algorithms and combinatorial optimization*, volume 2. Springer Science & Business Media, 2012.

21 D. Gusfield and R.W. Irving. *The Stable Marriage Problem: Structure and Algorithms*. Foundations of computing. MIT Press, 1989. URL: https://books.google.nl/books?id=2TzhSQAACAAJ.

22 R Halin. Lattices related to separation in graphs. In *Finite and Infinite Combinatorics in Sets and Logic*, pages 153–167. Springer, 1993.

23 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. doi:10.1609/AAAI.V37I4.25511.

24 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. doi:10.1609/AAAI.V35I5.16495.

25 Egbert Harzheim. *Ordered sets*, volume 7. Springer Science & Business Media, 2005.

26 Yuni Iwamasa, Tomoki Matsuda, Shunya Morihira, and Hanna Sumita. A general framework for finding diverse solutions via network flow and its applications. *arXiv preprint*, 2025. doi:10.48550/arXiv.2504.17633.

27 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. doi:10.1145/502090.502096.

28 Donald Ervin Knuth. *Stable marriage and its relation to other combinatorial problems: An introduction to the mathematical analysis of algorithms*, volume 10. American Mathematical Soc., 1997.

29 Soh Kumabe. Max-distance sparsification for diversification and clustering. *arXiv preprint*, 2024. doi:10.48550/arXiv.2411.02845.

30 George Markowsky. An overview of the poset of irreducibles. *Combinatorial And Computational Mathematics*, pages 162–177, 2001.

31 Bernd Meyer. On the lattices of cutsets in finite graphs. *European Journal of Combinatorics*, 3(2):153–157, 1982. doi:10.1016/S0195-6698(82)80028-0.

32 Neeldhara Misra, Harshil Mittal, and Ashutosh Rai. On the parameterized complexity of diverse sat. In *35th International Symposium on Algorithms and Computation (ISAAC 2024)*, pages 50:1–50:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.ISAAC.2024.50.

33 Neeraj Mittal and Vijay K Garg. Computation slicing: Techniques and theory. In *Distributed Computing: 15th International Conference, DISC 2001 Lisbon, Portugal, October 3–5, 2001 Proceedings 15*, pages 78–92. Springer, 2001. doi:10.1007/3-540-45414-4_6.

34 Kazuo Murota. *Discrete Convex Analysis*. Society for Industrial and Applied Mathematics, 2003. doi:10.1137/1.9780898718508.

35 Jean-Claude Picard and Maurice Queyranne. On the structure of all minimum cuts in a network and applications. *Math. Program.*, 22(1):121, December 1982. doi:10.1007/BF01581031.

36 Alexander Schrijver. A combinatorial algorithm minimizing submodular functions in strongly polynomial time. *Journal of Combinatorial Theory, Series B*, 80(2):346–355, 2000. doi:10.1006/JCTB.2000.1989.

37 Lloyd S Shapley and Martin Shubik. The assignment game I: The core. *International Journal of game theory*, 1(1):111–130, 1971.

38 Yuto Shida, Giulia Punzi, Yasuaki Kobayashi, Takeaki Uno, and Hiroki Arimura. Finding diverse strings and longest common subsequences in a graph. In *35th Annual Symposium on Combinatorial Pattern Matching (CPM 2024)*, pages 27:1–27:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.CPM.2024.27.

39 Marilda Sotomayor. The lattice structure of the set of stable outcomes of the multiple partners assignment game. *International Journal of Game Theory*, 28:567–583, 1999. doi:10.1007/S001820050126.

40 Richard P Stanley. Enumerative combinatorics: Volume 1, 2011.

Star-Based Separators for Intersection Graphs of c-Colored Pseudo-Segments

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

Bart M. P. Jansen
Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands

Jeroen S. K. Lamme
Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands

Abstract

The Planar Separator Theorem, which states that any planar graph $\mathcal{G}$ has a separator consisting of $O(\sqrt{n})$ nodes whose removal partitions $\mathcal{G}$ into components of size at most $\frac{2n}{3}$, is a widely used tool to obtain fast algorithms on planar graphs. Intersection graphs of disks, which generalize planar graphs, do not admit such separators. It has recently been shown that disk graphs *do* admit so-called clique-based separators that consist of $O(\sqrt{n})$ cliques. This result has been generalized to intersection graphs of various other types of disk-like objects. Unfortunately, segment intersection graphs do not admit small clique-based separators, because they can contain arbitrarily large bicliques. This is true even in the simple case of axis-aligned segments.

In this paper we therefore introduce *biclique-based separators* (and, in particular, *star-based separators*), which are separators consisting of a small number of bicliques (or stars). We prove that any c-oriented set of n segments in the plane, where c is a constant, admits a star-based separator consisting of $O(\sqrt{n})$ stars. In fact, our result is more general, as it applies to any set of n pseudo-segments that is partitioned into c subsets such that the pseudo-segments in the same subset are pairwise disjoint. We extend our result to intersection graphs of c-oriented polygons. These results immediately lead to an almost-exact distance oracle for such intersection graphs, which has $O(n\sqrt{n})$ storage and $O(\sqrt{n})$ query time, and that can report the hop-distance between any two query nodes in the intersection graph with an additive error of at most 2. This is the first distance oracle for such types of intersection graphs that has subquadratic storage and sublinear query time and that only has an additive error.

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

Keywords and phrases Computational geometry, intersection graphs, biclique-based separators, distance oracles

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.12

Related Version *Full Version*: https://arxiv.org/abs/2511.05371

Funding MdB and BJ are supported by the Dutch Research Council (NWO) through Gravitation-grant NETWORKS-024.002.003.

1 Introduction

Background. The celebrated Planar Separator Theorem by Lipton and Tarjan [24] states that any planar graph admits a balanced separator of size $O(\sqrt{n})$. More precisely, for any planar graph $G = (V, E)$ with n nodes there exists a set $S \subseteq V$ of size $O(\sqrt{n})$ whose removal partitions G into components of at most $\frac{2n}{3}$ nodes each. This fundamental tool has been used to develop efficient algorithms for many classic problems on planar graphs.

Geometric intersection graphs – graphs whose nodes correspond to objects in the plane and that have an edge between two nodes iff the corresponding objects intersect – are a generalization of planar graphs that have received widespread attention in computational

© Mark de Berg, Bart M.P. Jansen, and Jeroen S.K. Lamme;
licensed under Creative Commons License CC-BY 4.0

geometry, graph theory, and parametrized complexity. (For an overview of work in the latter area, we refer the reader to the survey by Xue and Zehavi [28].) Unfortunately, geometric intersection graphs do not admit balanced separators of sublinear size, because they can contain arbitrarily large cliques. This led De Berg et al. [11] to introduce so-called *clique-based separators*: balanced separators that consist of a small number of disjoint (but potentially large) cliques instead of a small number of nodes. They proved that any disk graph – and more generally, any intersection graph of convex fat objects in the plane – admits a clique-based separator of size $O(\sqrt{n})$. Here the *size* of a clique-based separator is the number of cliques it consists of. They also showed how to use such clique-based separators to obtain sub-exponential algorithms for various classic graph problems, including Independent Set, Dominating Set, and Feedback Vertex Set. Recently, De Berg et al. [12] showed that intersection graphs of pseudo-disks, and intersection graphs of geodesic disks inside a simple polygon, admit balanced separators consisting of $O(n^{2/3})$ cliques, and Aronov et al. [3] proved that intersection graphs of geodesic disks in any well-behaved metric in the plane admit balanced separators consisting of $O(n^{3/4+\varepsilon})$ cliques.

One may wonder if *all* geometric intersection graphs have sublinear clique-based separators. Unfortunately the answer is no, even for intersection graphs of horizontal and vertical line segments. The problem is that such graphs can contain arbitrarily large bicliques, and $K_{n,n}$ does not admit a sublinear clique-based separator. We therefore introduce *biclique-based separators*, which are separators consisting of bicliques, and we show that any set of horizontal and vertical segments admits a balanced separator consisting of a small number of bicliques. In fact, our result is stronger (as it uses *star graphs* in the separator, and not just any biclique) and it applies to a much wider class of intersection graphs, as discussed next.

Our contribution. Let $\mathcal{G} = (V, E)$ be an undirected graph with n nodes. A collection $S = \{S_1, \ldots S_t\}$ of (not necessarily induced) disjoint subgraphs from G is called a balanced[1] *biclique-based separator* for G if it has the following properties:

- Each subgraph S_i is a biclique.
- The removal from $\mathcal{G}$ of all subgraphs S_i and their incident edges partitions $\mathcal{G}$ into connected components with at most $\frac{2n}{3}$ nodes each.

The *size* of a biclique-based separator is the number of bicliques it is comprised of. If each biclique $S_i \in S$ is a star, then we call S a *star-based separator*. Note that the subgraphs S_i in a biclique-based separator (or: in a star-based separator) need not be induced subgraphs of $\mathcal{G}$. This is necessary to be able to handle large cliques. To avoid confusion between our new separators that are comprised of bicliques and the traditional separators that are comprised of individual nodes, we will refer to the latter as *node-based separators*.

We denote the intersection graph induced by a set V of n objects in the plane by $\mathcal{G}^\times[V]$. Thus, the nodes in $\mathcal{G}^\times[V]$ are in one-to-one correspondence with the objects in V and there is an edge between two objects $u, v \in V$ iff u intersects v. In Section 2 we prove that a star-based separator of size $O(\sqrt{n})$ exists for the intersection graph of any set V of axis-parallel segments. (The bound on the size of the separator is tight in the worst case.) In fact, we will prove that a star-based separator of size $O(\sqrt{n})$ exists for any set V of pseudo-segments[2] that is partitioned into subsets $V_1, \ldots, V_c$ such that the pseudo-segments from each V_i are disjoint from each other. In other words, each V_i is an independent set in $\mathcal{G}^\times[V]$. We call

[1] In the sequel we will often omit the adjective *balanced* and simply speak of *separators*.

[2] A set V of curves in the plane is a set of pseudo-segments if any two curves in V are either disjoint or intersect in a single point that is a proper crossing (and not a tangency).

such a set V a *c-colored set* of pseudo-segments, and we call $V_1, \ldots, V_c$ its color classes; see Figure 1(left). Note that a set of axis-parallel segments such that no two segments of the same orientation overlap, is a 2-colored set of pseudo-segments. More generally, a c-oriented set of line segments such that no two segments from the same orientation intersect, is a c-colored set of pseudo-segments. Intersection graphs of c-oriented segments are often referred to as c-DIR graphs, and when segments of the same orientation are not allowed to intersect, they are referred to as PURE c-DIR graphs [20, 21]. We show that a star-based separator of size $O(\sqrt{n})$ for a c-DIR graph can be computed in $O(n \log n)$ time, if the segments which induce this graph are given.

In Section 3 we extend our result to the case where V is a set of n constant-complexity c-oriented simple polygons, that is, a set of polygons without holes such that the set of edges of all polygons is a c-oriented set. Note that two polygons can intersect without having their boundaries intersect, namely when one polygon is completely contained in the other polygon – this is the main difficulty we need to handle when extending our results to polygons. Finally, in Section 3 we also present a straightforward greedy algorithm that computes a star-based separator of size $O(n^{2/3} \log^{2/3} n)$ for any string graph.

Application to distance oracles. A *distance oracle* for a (potentially weighted) graph $\mathcal{G} = (V, E)$ is a data structure that can quickly report the distance between two query nodes $s, t \in V$. Such queries can trivially be answered in $O(1)$ time if we store the distance between any two nodes in a distance table, but this requires $\Omega(n^2)$ storage. The challenge is to design distance oracles that use subquadratic storage. Unfortunately, this is not possible in general: any distance oracle must use $\Omega(n^2)$ bits of storage in the worst case, irrespective of the query time [27]. This is even true for distance oracles that approximate distances to within a factor strictly less than 3. Thus, work on distance oracles concentrated on special graph classes and, in particular, on planar graphs. More than two decades of research culminated in an exact distance oracle for weighted planar graphs that uses $O(n^{1+o(1)})$ storage and has $O(\log^2 n)$ query time [9]. For the unweighted case – in other words, if we are interested in the *hop-distance* – there is a $(1+\varepsilon)$-approximate distance oracle with $O(1/\varepsilon^2)$ query time and $O(n/\varepsilon^2)$ storage [22]. See the survey by Sommer [26] and the paper by Charalampopoulos et al. [9] for overviews of the existing distance oracles for various graph classes.

For geometric intersection graphs, only few results are known. Gao and Zhang [15], and Chan and Skrepetos [6], provide $(1+\varepsilon)$-approximate distance oracles with $O(n \log n)$ storage and $O(1)$ query time for weighted unit-disk graphs. No exact distance oracles that use subquadratic storage and have sublinear query time are known, even for unweighted unit-disk graphs. Very recently, Chang, Gao, and Le [8] presented an *almost exact* distance-oracle for unit-disk graphs (and, more generally, for intersection graphs of similarly sized, convex, fat pseudodisks) that uses $O(n^{2-1/18})$ storage and that can report the distance between two query nodes, up to an additive error[3] of 2, in $O(1)$ time. Another recent result is by Aronov, De Berg, and Theocharous [3], who presented an almost exact distance oracle that uses clique-based separators. For intersection graphs of geodesic disks in the plane, their oracle uses $O(n^{7/4+\varepsilon})$ storage, has $O(n^{3/4+\varepsilon})$ query time, and can report the hop-distance between two query points up to an additive error of 1. For Euclidean disks, the storage and preprocessing would be $O(n\sqrt{n})$ and $O(\sqrt{n})$, respectively. As we explain later, their approach also works with biclique-based separators; the only difference is that the additive

[3] In the most recent arxiv version [7] the error has been reduced to 1.

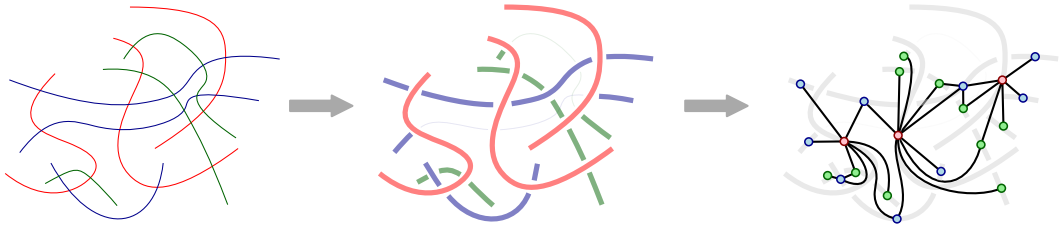


Figure 1 Left: A 3-colored set of pseudo-segments. Middle: The active fragments created by our algorithm. Right: The contact graph induced by the active fragments.

error increases from 1 to 2. Thus, we obtain an almost exact distance oracle for intersection graphs of c-colored pseudo-segments with $O(n\sqrt{n})$ storage and $O(\sqrt{n})$ query time. This is, to the best of our knowledge, the first almost exact distance oracle for intersection graphs of non-disk-like objects.

2 A star-based separator for c-colored pseudo-segments

Let V be a c-colored set of pseudo-segments, as defined above. To simplify the terminology, from now on we simply refer to the pseudo-segments in V as *segments*. We assume that the segments in V are in general position and, in particular, that no three segments meet in a common point and that no endpoint of one segment lies on another segment. This assumption is without loss of generality, as it can always be ensured by perturbing the segments slightly.

The construction. Recall that a *contact graph* is the intersection graph of a set of interior-disjoint objects. Contact graphs of curves are known to be planar if no four objects meet in a common point [19, Lemma 2.1]. Our strategy to construct a star-based separator S for $\mathcal{G}^{\times}[V]$ consists of the following steps, illustrated in Figure 1 and explained in more detail later.

Step 1. We partition each segment in V into *fragments*. Some fragments will be *active*, while others will be *inactive*. This partition will be such that active segments do not cross each other, although they may touch.

Step 2. Let $\mathcal{H}$ be the contact graph on the active segments. We construct a separator $S_{\mathcal{H}}$ on $\mathcal{H}$, using a suitable weighting scheme on the nodes of $\mathcal{H}$. Because $\mathcal{H}$ is planar, constructing the separator can be done using the Planar Separator Theorem.

Step 3. We use $S_{\mathcal{H}}$ to construct our star-based separator S for $\mathcal{G}^{\times}[V]$. For a fragment f, let $\text{seg}(f) \in V$ denote the segment containing f. Intuitively, we want to put a star into S for each fragment f in the separator $S_{\mathcal{H}}$, namely, the star consisting of the segment $\text{seg}(f)$ as well as all other segments intersecting $\text{seg}(f)$. For technical reasons, however, we actually have to put a slightly larger collection of stars into S.

To make this strategy work, we need to control the size of the contact graph $\mathcal{H}$. More precisely, to obtain a star-based separator of size $O(\sqrt{n})$, the size of $\mathcal{H}$ needs to be $O(n)$. Thus we cannot, for example, cut each segment into fragments at its intersection points with all other segments and make all the resulting fragments active. On the other hand, if we ignore certain parts of the segments by making them inactive, we miss certain intersections and we run the risk that our final set of stars is no longer a valid separator. Next, we describe how to overcome these problems by carefully creating the active fragments.

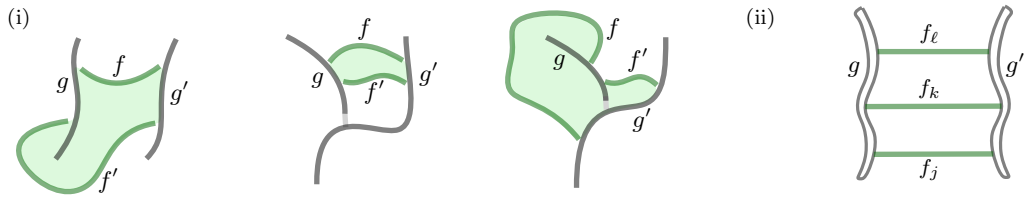


Figure 2 (i) Defining the region $Q(f, f', g, g')$. (ii) Illustration for the proof of Lemma 2.

Step 1: Creating the active fragments. To construct the active fragments that form the nodes in our contact graph $\mathcal{H}$, we will go over the subsets $V_1, \ldots, V_c$ one by one. We denote the active and inactive fragments created for a subset V_i by F_i and $\overline{F}_i$, respectively. For $1 \leqslant i \leqslant c$, we define $F_{\leqslant i} := F_1 \cup \cdots \cup F_i$, and we define $F_{<i}$ similarly.

Handling the first subset V_1 is easy: we simply define $F_1 := V_1$. In other words, each segment in V_1 becomes a single fragment, and all these fragments are active.

Now consider a subset V_i with $i > 1$. Each segment $v \in V_i$ is partitioned into one or more fragments by cutting it at every intersection point of v with an active fragment $f \in F_{<i}$. Let X_i be the set of fragments thus created. There are two types of fragments in X_i: fragments f that contain an endpoint of the segment $v \in V_i$ contributing f – there are at least one and at most two of these fragments per segment $v \in V_i$ – and fragments that do not contain such an endpoint. We call fragments of the former type *end fragments* and fragments of the latter type *internal fragments*. Note that an internal fragment has its endpoints on two distinct active fragments $g, g' \in F_{<i}$. We then say that f *connects* g and g'.

Now that we have defined X_i, we need to decide which fragments in X_i become active. To avoid making too many fragments active, we will partition X_i into equivalence classes, and we will activate only one fragment from each equivalence class. To define the equivalence classes we first define, for two internal fragments $f, f' \in X_i$ that connect the same pair of fragments $g, g' \in F_{<i}$, a region $Q(f, f', g, g')$, as follows; see Figure 2(i) for an illustration.

First, suppose that the segments g and g' do not touch each other, as in the left part of Figure 2(i). Thus, $\mathbb{R}^2 \setminus (g \cup g')$ is a single, unbounded region with two holes, namely g and g'. The fragment f connects these two holes, and so $\mathbb{R}^2 \setminus (g \cup g' \cup f)$ is still a single unbounded region, but now with one hole. Removing f' from this region splits it into two regions, one bounded and one unbounded. We define $Q(f, f', g, g')$ to be the bounded region. Now suppose that g and g' touch each other, say at an endpoint of g. We slightly shrink g at the point where it touches g', and then define $Q(f, f', g, g')$ as above. Note that in this case $Q(f, f', g, g')$ may consist of one or two bounded regions, if we undo the shrinking process; see the middle and right part in Figure 2(i). We can now define the equivalence classes.

▶ **Definition 1.** *Let f and f' be two fragments in X_i. We say that f and f' are* equivalent, *denoted by $f \equiv f'$, if $f = f'$ or the following two conditions hold.*

- **(i)** *The fragments f and f' are internal and connect the same pair of fragments $g, g' \in F_{<i}$.*
- **(ii)** *The region $Q(f, f', g, g')$ enclosed by the fragments f, f', g, g' does not contain an endpoint of any segment in V.*

The following lemma shows that $\equiv$ is indeed an equivalence relation.

▶ **Lemma 2.** *The relation $\equiv$ defined in Definition 1 is an equivalence relation.*

Proof. It is clear that $\equiv$ is reflexive and symmetric. It remains to show that if $f_1 \equiv f_2$ and $f_2 \equiv f_3$, then $f_1 \equiv f_3$. Let g, g' be the fragments connected by f_1, f_2, f_3. We can assume that g and g' do not touch; otherwise, as before, we can shrink one of the fragments so that

the arguments below still apply. It is instructive to view g and g' as being slightly inflated so that they become closed curves and no longer have two "sides". We then see that, from a topological point of view, the situation is always as in Figure 2(ii): two of the fragments, f_j and f_ℓ, are incident to the unbounded face, while the third fragment f_k is not. Thus, if we define $Q_{jk} := Q(f_j, f_k, g, g')$ and we define $Q_{k\ell}$ and $Q_{j\ell}$ similarly, then $Q_{j\ell} = Q_{jk} \cup Q_{k\ell}$. This implies that, no matter which of the three fragments f_j, f_k, f_ℓ is f_2, we always have $Q_{13} \subseteq Q_{12} \cup Q_{23}$. Since Q_{12} and Q_{23} do not contain endpoints of segments in V if $f_1 \equiv f_2$ and $f_2 \equiv f_3$, neither does Q_{13}. Thus, $f_1 \equiv f_3$. ◀

We now partition X_i into equivalence classes according to the relation $\equiv$ defined above. For each equivalence class, we make an arbitrary fragment $f \in X_i$ from that class active and put it into F_i; the other fragments from that equivalence class are made inactive and put into $\overline{F}_i$. Note that end fragments are always active, since they do not connect a pair of fragments from $F_{<i}$ and thus cannot be equivalent to any other fragment.

Step 2: Creating the contact graph $\mathcal{H}$ and its separator $S_\mathcal{H}$. Let $F := F_1 \cup \cdots \cup F_c$ be the set of active fragments created in Step 1, and let $\overline{F} := \overline{F}_1 \cup \cdots \cup \overline{F}_c$ be the inactive fragments. We define $\mathcal{H} = (F, E_\mathcal{H})$ to be the contact graph of F. More precisely, for two fragments $f, f \in F$ we put the edge (f, f') in $E_\mathcal{H}$ iff f and f' are in contact – that is, $f \cap f' \neq \emptyset$ – and they do not belong[4] to the same segment in V; see Figure 1. Because of our general position assumption, no three fragments from different segments meet in a point, and therefore $\mathcal{H}$ is planar [19, Lemma 2.1]. From now on, with a slight abuse of notation, we will not make a distinction between the nodes in $\mathcal{H}$ and the corresponding fragments in F.

We now wish to create a separator $S_\mathcal{H}$ for $\mathcal{H}$. In Step 3 we will use $S_\mathcal{H}$ to create a separator S for $\mathcal{G}^\times[V]$. To ensure that S will be balanced, we will put weights on the nodes in $\mathcal{H}$ and use a weighted version of the Planar Separator Theorem, as described next.

For each segment $s \in V$, we designate one of its end fragments – recall that end fragments are always active – as its *representative fragment*. We give all representative fragments a weight of $\frac{1}{n}$, and all other fragments a weight of 0. Note that the total weight of the fragments in F is 1. We apply the weighted separator theorem given below to $\mathcal{H}$. This gives us a separator $S_\mathcal{H}$, and parts $A_\mathcal{H}, B_\mathcal{H} \subseteq F \setminus S_\mathcal{H}$ such that there are no edges between $A_\mathcal{H}$ and $B_\mathcal{H}$.

▶ **Lemma 3** (Theorem 4 in [24]). *Let G be a non-negatively weighted planar graph containing n nodes whose weights sum up to at most 1. The node set V_G can be partitioned into a separator S, and sets A and B such that $|S| = O(\sqrt{n})$, no edge connects A and B, and the total weight of the nodes in A, as well as the total weight of the nodes in B, is at most $\frac{2}{3}$.*

Step 3: Creating the star-based separator S for $\mathcal{G}^\times[V]$. Using the separator $S_\mathcal{H}$ created in Step 2, we now create our star-based separator S for the intersection graph $\mathcal{G}^\times[V]$. We do this by putting one or three stars into S for each fragment $f \in S_\mathcal{H}$, as follows. For a segment $s \in V$, define $\text{star}(s)$ to be the subgraph of $\mathcal{G}^\times[V]$ consisting of s and all its incident edges. Thus, the nodes in $\text{star}(s)$ are the segment s itself plus the segments $s' \in V$ that intersect s.

- If $f \in S_\mathcal{H}$ is an end fragment then we put $\text{star}(\text{seg}(f))$ into S.
- If $f \in S_\mathcal{H}$ is an internal fragment then let $g, g' \in F$ be the pair of active fragments connected by f. We put $\text{star}(\text{seg}(f))$, $\text{star}(\text{seg}(g))$, and $\text{star}(\text{seg}(g'))$ into S.

[4] Since we do not put an edge between fragments belonging to the same segment, even if these fragments touch, $\mathcal{H}$ is formally speaking not a contact graph, but we permit ourselves this abuse of terminology.

Note that multiple copies of the same star can be added to S. We remove these duplicates to ensure that all star graphs in S have unique centers. It can still be the case that several stars in S contain the same node. To make the stars pairwise disjoint, we therefore remove non-center nodes until every node appears in at most one star in S.

The analysis. We now show that the construction described above yields a balanced separator of the required size. This requires proving two things: the *separation property*, namely that the removal of S partitions $\mathcal{G}^\times[V]$ into components of size at most $\frac{2n}{3}$, and the *size property*, namely that S consists of $O(\sqrt{n})$ stars.

Proving the separation property. To prove the separation property, it suffices to show that $V \setminus S$ can be partitioned[5] into subsets A and B such that $|A| \leqslant \frac{2n}{3}$ and $|B| \leqslant \frac{2n}{3}$, and such that no segment in A intersects any segment in B.

We define the sets A and B as follows. For each segment $v \in V$ not contained in a star in S we look at its representative fragment f_v. Note that f_v must be contained in either $A_\mathcal{H}$ or $B_\mathcal{H}$, since $f_v \in S_\mathcal{H}$ would imply that v is contained in a star in S. If $f_v \in A_\mathcal{H}$ then we add v to A, else we add v to B. The value $\frac{|A|}{n}$ can be at most the total weight of fragments in $A_\mathcal{H}$, and a similar statement holds for B. Hence, the next observation follows from the fact that $S_\mathcal{H}$ is a balanced separator.

▶ **Observation 4.** *$|A| \leqslant \frac{2n}{3}$ and $|B| \leqslant \frac{2n}{3}$.*

The more challenging part is to show that no segment in A intersects any segment in B. We will need the following lemma. Recall that the segment set V is partitioned into color classes $V_1, \ldots, V_c$, which we handled one by one to create the set F of active fragments.

▶ **Lemma 5.** *Let $v \in V_i \setminus S$ and let f be an active fragment. Suppose one of the following conditions holds:*

(i) *$f \in F_{<i}$ and v intersects f,*
(ii) *$\mathrm{seg}(f) = v$, or*
(iii) *f is equivalent to an inactive fragment f' such that $\mathrm{seg}(f') = v$.*

Then $f \in A_\mathcal{H}$ if $v \in A$, and $f \in B_\mathcal{H}$ if $v \in B$.

Proof. We prove the lemma under the assumption that $v \in A$; the proof for $v \in B$ is analogous. Let f_v be the representative fragment of v. Because $v \in A$, we have $f_v \in A_\mathcal{H}$. We will define sets Z_1, Z_2, Z_3 that contain the active fragments for which conditions (i), (ii), and (iii) hold, respectively, and then argue that the lemma holds for each of the three sets.

Let f'_v be the other end fragment of v, and let $Z_1 = \{g_1, \ldots, g_k\}$ be the ordered set of fragments from $F_{<i}$ that we cross as we trace v from f_v to f'_v; see Figure 3. It is possible that f'_v does not exist, in this case Z_1 is empty. Note that every pair $g_j, g_{j+1} \in Z_1$ is connected by an active or inactive fragment of v, which we denote by f_j. We now define Z_2, Z_3 as follows.

- Z_2 contains the fragments f_j that are active plus the end fragments f_v and (if it exists) f'_v. Thus, Z_2 simply contains all active fragments of v.
- Z_3 contains, for each inactive fragment f_j, the unique equivalent active fragment $f'_j \in F_i$.

It is easily checked that the sets Z_1, Z_2, Z_3 indeed contain exactly those fragments for which conditions (i), (ii), and (iii) hold, respectively. We will now prove that all fragments in $Z_1 \cup Z_2 \cup Z_3$ are in $A_\mathcal{H}$. To this end, observe that the fragments in $Z_1 \cup Z_2 \cup Z_3$ correspond

[5] Formally, we should have written $V \setminus \bigcup S$ instead of $V \setminus S$, since $S_\mathcal{H}$ is a set of stars and not a set of nodes, but we prefer the simpler (though technically incorrect) notation.

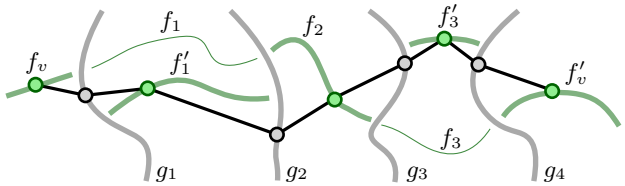


Figure 3 The various fragments used in the proof of Lemma 5. Note that the segments $\text{seg}(f'_1)$ and $\text{seg}(f'_3)$ are not drawn in their entirety. The path in $\mathcal{H}$ from f_v to f'_v is also shown.

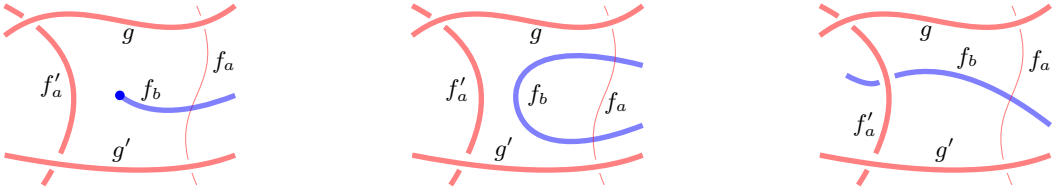


Figure 4 Left: The segment b has an endpoint in Q. Middle: The segment b intersects a twice. Right: The segment b intersects f'_a. All of these cases lead to a contradiction.

to nodes in $\mathcal{H}$ that form a path π starting at f_v and ending at f'_v, as illustrated in Figure 3. Recall that $f_v \in A_{\mathcal{H}}$. Now assume for a contradiction that there is a fragment $f \in Z_1 \cup Z_2 \cup Z_3$ that is in $B_{\mathcal{H}} \cup S_{\mathcal{H}}$. Because there are no edges between $A_{\mathcal{H}}$ and $B_{\mathcal{H}}$, this means there must be a fragment f^* on the subpath of π from f_v to f that is an element of the separator $S_{\mathcal{H}}$. If $f^* \in Z_1 \cup Z_2$ then $v \in \text{star}(\text{seg}(f^*))$, which contradicts that $v \in A$. Otherwise $f^* \in Z_3$ and f^* connects two fragments $g_j, g_{j+1} \in Z_1$. Since $f^* \in S_{\mathcal{H}}$, this implies that $\text{star}(\text{seg}(g_j))$ is a star in S. Because $v \in \text{star}(\text{seg}(g_j))$, this again contradicts that $v \in A$. ◀

We can now prove that $\mathcal{G}^\times[V]$ does not contain edges between nodes in A and nodes in B.

▶ **Lemma 6.** *No segment $a \in A$ intersects any segment $b \in B$.*

Proof. Assume for a contradiction that $a \in A$ and $b \in B$ intersect. Let $f_a \subseteq a$ and $f_b \subseteq b$ be the fragments containing the intersection point. We distinguish three cases.

- *Both f_a and f_b are active.* Then f_a and f_b satisfy condition (ii) of Lemma 5, and so $f_a \in A_{\mathcal{H}}$ and $f_b \in B_{\mathcal{H}}$. Because f_a and f_b are active and intersect, (f_a, f_b) is an edge in $\mathcal{H}$. But then $S_{\mathcal{H}}$ would not be a proper separator, and we reach a contradiction.
- *Both f_a and f_b are inactive.* Let $f_a \in \overline{F}_i$ and $f_b \in \overline{F}_j$, and assume without loss of generality that $i \leqslant j$. Because a and b intersect and segments from the same color class are disjoint, we cannot have $i = j$. Hence, $i < j$. Because f_a is inactive, it must be an internal fragment that connects some fragments $g, g' \in F_{<i} \subseteq F_{<j}$. Thus, g, g' satisfy condition (i) of Lemma 5, with a playing the role of v, which implies that $g, g' \in A_{\mathcal{H}}$. Let $f'_a \in F_i$ be the active fragment that is equivalent to f_a. Then f'_a satisfies condition (iii) from Lemma 5 and so $f'_a \in A_{\mathcal{H}}$. The fragments f_a, f'_a, g, g' enclose some region Q. The segment b intersects f_a so it is partly contained in Q; see Figure 4 for an illustration of the possibilities for f_b entering Q. It cannot have an endpoint in Q, because f_a and f'_a are equivalent. Segment b cannot intersect f_a twice, because V is a collection of pseudo-segments. It follows that b must intersect f_a, g or g'. Now observe that f_a, g and g' are all fragments in $F_{<j}$. This implies that the fragment that intersects b satisfies condition (i) from Lemma 5, with b playing the role of v. But then the intersected fragment would be in $B_{\mathcal{H}}$, contradicting that $A_{\mathcal{H}}$ and $B_{\mathcal{H}}$ are disjoint.

- *One of the fragments f_a, f_b is active and one is inactive.* Assume wlog that f_a is inactive and f_b is active. Let $f_a \in \overline{F}_i$ and $f_b \in F_j$. Because a and b intersect, we have $i \neq j$. If $i < j$ then the arguments from the previous case can be used to obtain a contradiction – indeed, these arguments did not use that f_b is inactive. If $i > j$ then $f_b \in F_{<i}$. Also note that a intersects f_b. But then f_b satisfies condition (i) from Lemma 5, with a playing the role of v. It follows that $f_b \in A_{\mathcal{H}}$, which contradicts that $A_{\mathcal{H}}$ and $B_{\mathcal{H}}$ are disjoint.

We have reached a contradiction in each case, thus proving the lemma. ◀

Proving the size property. Because we add at most three stars to S per fragment in $S_{\mathcal{H}}$, it suffices to bound the size of $S_{\mathcal{H}}$ to prove the size property. From Lemma 3 it follows that $|S_{\mathcal{H}}| = O(\sqrt{|F|})$. The next lemma bounds $|F|$.

▶ **Lemma 7.** *$|F| = O(n \cdot 4^c)$, where c is the number of color classes in V.*

Proof. We first bound $|F_i|$, the number of active fragments created for the segments in V_i, in terms of the number of active fragments created for $V_{<i}$.

▷ Claim. For $1 < i \leqslant c$ we have $|F_i| \leqslant 2 \cdot |V_i| + 3 \cdot (|F_{<i}| + 2n + 1) - 6$.

Proof. The set F_i contains at most $2 \cdot |V_i|$ end fragments. The internal fragments in F_i connect two fragments from $F_{<i}$. We denote the set of these internal fragments by F_i^{int}. Now consider the multi-graph $\mathcal{G}$ defined as follows.

- For each fragment in $F_{<i}$, we add a node to $\mathcal{G}$.
- For each fragment in F_i^{int}, we add an edge to $\mathcal{G}$ between the fragments from $F_{<i}$ that it connects.
- For each endpoint of a segment in $V_{\geqslant i}$ we add a singleton node to $\mathcal{G}$. (Observe that the endpoints of segments in $V_{<i}$ lie on an end fragment in $F_{<i}$, which is already a node in $\mathcal{G}$.)

Now consider the obvious drawing of $\mathcal{G}$, where the nodes are drawn as fragments of $F_{<i}$ or as points, and the edges are drawn as fragment in F_i^{int}. Recall that two active fragments can touch, but they never cross. By continuously shrinking the fragments in $F_{\leqslant i}$ and deforming the fragments of F_i^{int} appropriately, we can therefore create a plane drawing of $\mathcal{G}$. That is, we can create a drawing of $\mathcal{G}$ in which the nodes are points and the edges are pairwise disjoint curves connecting their endpoints. See Figure 5 for an example of this deformation. For reasons that will become clear shortly, we augment $\mathcal{G}$ with one additional singleton node u_∞, which we place in the unbounded face of $\mathcal{G}$.

The graph $\mathcal{G}$ is a multi-graph because F_i^{int} can contain multiple fragments connecting the same pair of fragments $f, f' \in F_{<i}$. Let $g, g' \in F_i^{\text{int}}$ be two such fragments. The reason that we added both g and g' to F_i^{int} is that g and g' were not equivalent. Hence, the region $Q(f, f', g, g')$ contains an endpoint belonging to some segment $v \in V$. The deformation process that turns each node the drawing of $\mathcal{G}$ into a point can be done in such a way that this property is maintained. Thus, after the deformation we have a plane drawing of $\mathcal{G}$ in which for any two edges g, g' that connect the same pair of nodes, there is a node inside the deformed region $Q(f, f', g, g')$. Because of the additional node u_∞, we are also guaranteed to have at least one node outside this region. A plane multi-graph with this property is called a *thin graph*. It is known [1, Lemma 5] that the standard inequality $(\#\text{edges}) \leqslant 3 \cdot (\#\text{nodes}) - 6$ that holds for planar graphs (with at least three vertices) also holds for thin graphs. Hence, $|F_i^{\text{int}}| \leqslant 3(|F_{<i}| + 2n + 1) - 6$. ◁

Note that $|F_1| = |V_1|$ and $|V_i| \leqslant n$ for all i. Hence, the claim above gives us the recurrence $|F_{\leqslant i}| \leqslant 4|F_{<i}| + 8n - 3$ with $|F_{\leqslant 1}| \leqslant n$. This gives $|F_{\leqslant i}| \leqslant \left(\frac{n}{4} + \frac{8n-3}{12}\right) \cdot 4^i - \frac{8n-3}{3}$. Plugging in $i = c$ gives $|F| = |F_{\leqslant c}| = O(n \cdot 4^c)$, which proves the lemma. ◀

Figure 5 Left: Fragments in V_i are green, fragments in F_i^{int} are dark gray, and endpoints of segments in $V_{\geqslant i}$ are blue. Right: The plane multigraph created for the example on the left.

Since c, the number of color classes, is a constant, we obtain the following corollary.

▶ **Corollary 8.** *The separator S contains $O(\sqrt{n})$ star graphs.*

So far we have considered input sets V where the segments are unweighted. To create a separator for c-oriented polygons, which we will do in the next section, we need a separator for weighted segments. Fortunately it is straightforward to adapt the construction described above to the weighted setting – we only need to change the weighting scheme we used in Step 2 of the construction. More precisely, instead of assigning a weight $\frac{1}{n}$ to the representative fragment of a segment v, we assign $\text{weight}(v)/\sum_{u\in V}\text{weight}(u)$ to the representative.

▶ Remark. It is well known that grid graphs do not admit node-based separators of size $o(\sqrt{n})$. Because nodes in grid graphs have constant degree, bicliques in grid graphs have constant size. Hence, grid graphs do not admit biclique-based separators of size $o(\sqrt{n})$. Grid graphs are bipartite and planar, which implies that they are PURE-2-DIR graphs [18]. We conclude that even PURE-2-DIR graphs do not admit biclique based separators of size $o(\sqrt{n})$.

Computation time. If we assume that the appropriate elementary operations on the pseudo-segments – computing the intersection point of two pseudo-segments, for instance, or determining if a point lies inside some region $Q(f, f', g, g')$ – can be performed in $O(1)$ time, that then a brute-force implementation of the algorithm presented above runs in polynomial time. More interestingly, for c-oriented line segments, the algorithm can be implemented to run in $O(n \log n)$ time, as shown in the full version of this paper. We obtain the following theorem.

▶ **Theorem 9.** *Let V be a c-colored set of n non-negatively weighted pseudo-segments, where c is a fixed constant, whose total weight is at most 1. Then the intersection graph $\mathcal{G}^\times[V]$ has a star-based separator S of size $O(\sqrt{n})$ such that $V \setminus S$ can be partitioned into subsets A, B of weight at most $\frac{2}{3}$ with no edges between them. The bound on the size of the separator is tight, even for axis-parallel segments. In the special case where V is a set of c-oriented line segments, the separator S and parts A, B can be computed in $O(n \log n)$ time.*

Application to distance oracles. Arikati et al. [2] presented a simple distance oracle for planar graphs, using node-based separators. Aronov, De Berg, and Theocharous [3] observed that the approach can be adapted to work with clique-based separators, as follows. Let $\mathcal{G} = (V, E)$ be the graph for which we want to construct a distance oracle.

- Construct a clique-based separator S for $\mathcal{G}$, and let $A, B \subseteq V \setminus S$ be the two parts of the partition given by S. For each node $v \in V$ and each clique $C \in S$, store the distance $d(v, C) := \min\{d(v, u) : u \in C\}$, where $d(u, v)$ denotes the hop-distance from s to t in $\mathcal{G}$.
- Recursively construct distance oracles for the subgraphs induced by the parts A and B.

Now suppose we want to answer a distance query with nodes $s, t \in V$. Let $d^* := \min\{d(s, C) + d(t, C) : C \in S\}$. If s and t do not lie in the same part – that is, we do not have $s, t \in A$ or $s, t \in B$ – then we report d^*. Otherwise, s and t lie in the same part of the partition, say A. Then we report the minimum of d^* and the distance we obtain by querying the recursively constructed oracle for A.

This distance oracle uses $O(n \cdot s(n))$ storage, where $s(n)$ is the size of the separator, and it has $O(s(n))$ query time, assuming $s(n) = \Omega(n^\beta)$ for some constant $\beta > 0$. The reported distance is either the exact distance $d(s, t)$, or it is $d(s, t) - 1$. The additive error of 1 is because we do not know if s and t can reach the same node of some clique C with paths of length $d(s, C)$ and $d(t, C)$, respectively – we may have to use an edge inside C to connect these paths. We observe that the same approach can be used in combination with star-based (or biclique-based) separators. The only difference is that we now get an additive error of at most 2, because we may need two additional edges inside a star (or biclique) in the separator. We obtain the following result.

▶ **Corollary 10.** *Let V be a c-colored set of pseudo-segments, where c is a fixed constant. There is an almost-exact distance oracle for $\mathcal{G}^\times[V]$ that uses $O(n\sqrt{n})$ storage and can report the hop-distance between any two nodes $s, t \in V$, up to an additive error of 2, in $O(\sqrt{n})$ time.*

3 Extension to c-oriented polygons and string graphs

c-Oriented polygons. Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be a collection of c-oriented simple polygons, each with a constant number of edges, where c is a fixed constant. The idea is to create a weighted collection V of segments to which we can apply Theorem 9, and then use the resulting separator S_V to construct a separator S for $\mathcal{G}^\times[\mathcal{P}]$. The set V is created as follows.

- First, we add each side of every polygon $P_i \in \mathcal{P}$ to V. For each polygon P_i, we pick an arbitrary side as its *representative side*, which we give weight $\frac{1}{n}$; other sides of P_i are given weight 0.
- Second, to handle the containment of polygons within other polygons, we add so-called *containment segments* to V. These segments will always have weight 0. Let ϕ be an orientation that is not used by any segment in V. For each polygon P_i let $\mathcal{C}_i \subseteq \mathcal{P}$ be the set of polygons that fully contain P_i. Take a point $x_i \in P_i$, and let ρ_i be a ray emanating from x_i and with orientation ϕ. For each $P_j \in \mathcal{C}_i$, let y_j be the point where ρ_i leaves P_j for the first time. Define x_i' to be the last such point y_{j^*}. We now add $x_i x_i'$ as a containment segment to V. Note that the containment segment $x_i x_i'$ intersects the boundaries of all polygons in $\mathcal{C}_i$. Moreover, $x_i x_i'$ is completely contained in P_{j^*}. Figure 6 shows examples of containment segments.

Clearly, we can partition V into color classes $V_1, \ldots, V_{c+1}$ based on the orientation of the segments. We let V_1 contain the segments of orientation ϕ, which will be important later. The total weight of the segments is 1, so we can apply Theorem 9. Let S_V be the resulting separator for $\mathcal{G}^\times[V]$, and let A_V and B_V be the two parts of weight at most $\frac{2}{3}$ into which S_V splits $V \setminus S_V$. We construct a separator S and parts A, B for $\mathcal{G}^\times[\mathcal{P}]$ as follows.

For each star in S_V we consider its center v. If v is a side of a polygon $P_i \in \mathcal{P}$ then we add $\text{star}(P_i)$ to S, where $\text{star}(P_i)$ is the subgraph of $\mathcal{G}^\times[\mathcal{P}]$ consisting of P_i and its incident edges. If v is a containment segment $x_i x_i'$ that we generated for polygon $P_i \in \mathcal{P}$, then let P_j be the polygon enclosing P_i such that $x_i' \in \partial P_j$. We add $\text{star}(P_j)$ to S_P. As before, we remove duplicate stars and we remove polygons from stars to ensure that each polygon is in at most one star. To create the parts A, B, we consider the representative sides of the polygons $P_i \in \mathcal{P}$ that are not in a star in S. If the representative side is in A_V, then we put P_i in part A; otherwise we put P_i in part B.

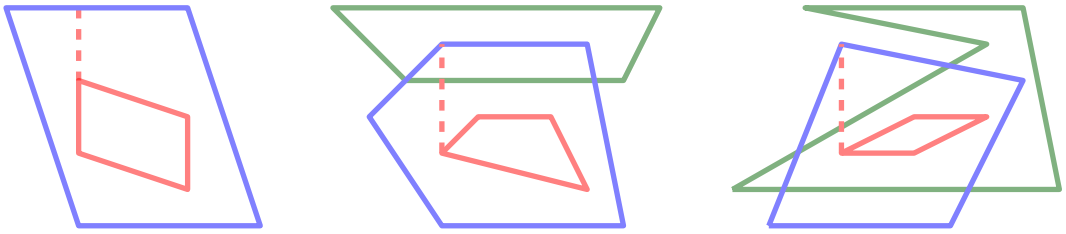


Figure 6 Left: The dashed line is the containment segment for the red polygon. Middle: The containment segment for the red polygon intersects a polygon which does not contain the red polygon, this is allowed. Right: The containment segment stops after intersecting the boundary of every polygon which contains the red polygon. Extending the containment segment could lead to further intersections with the green polygon. However, only the first intersection between the containment segment and a polygon are considered in the construction.

The analysis. Proving the size property, namely that $|S| = O(\sqrt{n})$, is easy. Indeed, we put $O(1)$ sides and one containment segment per polygon into V. Hence, $|V| = O(n)$ and thus $|S| = |S_V| = O(\sqrt{n})$. It remains to prove the separation property.

Since the total weight of A_V and of B_V are both at most $\frac{2}{3}$ and each representative side has weight $\frac{1}{n}$, it follows that $|A| \leqslant \frac{2n}{3}$ and $|B| \leqslant \frac{2n}{3}$. Next, we prove that there are no edges from A to B in $\mathcal{G}^{\times}[\mathcal{P}]$. We need the following lemma.

▶ **Lemma 11.** *Consider a polygon $P_i \in \mathcal{P}$. If $P_i \in A$ (resp. $P_i \in B$), then all sides of P_i are in A_V (resp. B_V).*

Proof. First consider the case $P_i \in A$. Suppose for a contradiction that not all sides of P_i are in A_V. Hence, P_i has a side $s \in S_V$ or a side $s' \in B_V$. We claim that in the latter case P_i must also have a side $s \in S_V$. To see this, consider the representative side s_i of P_i. Note that $s_i \in A_V$ since we put P_i into A. Also note that there is a path in $\mathcal{G}^{\times}[V]$ connecting s' to s_i – the nodes on this path correspond to the sides of P_i that we encounter as we traverse ∂P_i from s' to s_i in, say, clockwise order. Since $s_i \in A_V$ and $s' \in B_V$, one of the nodes on this path must be in S_V and be a side of P_i. This establishes our claim.

It remains to prove that if P_i has a side s in S_V, then $P_i \in S$. If s is the center of a star in S_V, then by construction P_i is the center of a star in S. If s is a non-center node in some $\text{star}(s') \in S_V$ we distinguish two cases. If s' is a side of some polygon P_j, then P_j intersects P_i and S contains $\text{star}(P_j)$. Consequently, P_i is a non-center node in $\text{star}(P_j)$, thus $P_i \in S$. If s' is a containment segment $x_j x'_j$ where x'_j lies on some polygon P_k, then P_k intersects P_i because $x_j x'_j$ is fully contained in P_k. Because $\text{star}(s') \in S_V$ it follows that $\text{star}(P_k) \in S$, and thus $P_i \in S$. We have reached a contradiction in all cases. We conclude that all sides of P_i are in A_V.

Now consider the case $P_i \in B$. We can follow the proof for the case $P_i \in A$ if the representative side s_i of P_i is in B_V. This must indeed be the case: we cannot have $s_i \in A_V$ because then we would have put P_i into A, and s_i cannot be a node in a star in S_V because then P_i would have been in a star in S, as has been shown above. Hence, the lemma is also true if $P_i \in B$. ◀

We can now prove that there are no edges from A to B in $\mathcal{G}^{\times}[\mathcal{P}]$.

▶ **Lemma 12.** *No polygon $P_a \in A$ intersects any polygon $P_b \in B$.*

Proof. Suppose for a contradiction that P_a and P_b intersect. We distinguish two cases.

- *The boundaries of P_a and P_b intersect.* Let s and s' be the sides of P_a and P_b that intersect. Because $P_a \in A$ we have $s \in A_V$ by Lemma 11. Similarly, we have $s' \in B_V$. But this contradicts that S_V is a separator for $\mathcal{G}^\times[V]$ with parts A_V, B_V.
- *The boundaries of P_a and P_b do not intersect.* Assume wlog that $P_a \subset P_b$. Consider the containment segment $s = x_a x'_a$. By construction of the containment segments, polygon P_b has a side s' that intersects s. Recall that the containment segments were put into the set V_1 that was handled first by the algorithm from the previous section. Hence, there will be a fragment $f \in F$ that is identical to s. Let $i > 1$ be such that $s' \in V_i$. Then f satisfies condition (i) of Lemma 5, with s' playing the role of v. Since s' is a side of $P_b \in B$, we have $s' \in B_V$ by Lemma 11. Thus, Lemma 5 implies that $s = f \in B_V$.
  Similarly there exists a side s'' that is part of P_a that intersects s. From Lemma 11 it follows $s'' \in A_V$. It must be that $s'' \in V_j$ for some $j > 1$. Then f also satisfies condition (i) of Lemma 5 with s'' playing the role of v, so that $s = f \in A_V$. But then the sets A_V and B_V are not disjoint, which contradicts that S_V is a separator. ◀

Putting everything together, we obtain the following theorem. The runtime guarantee for computing the separator is proven in the full version.

▶ **Theorem 13.** *Let P be a set of n constant-complexity c-oriented polygons in the plane. Then the intersection graph $\mathcal{G}^\times[P]$ has a star-based separator of size $O(\sqrt{n})$, which can be computed in $O(n \log n)$ time. Moreover, there is an almost exact distance oracle for $\mathcal{G}^\times[V]$ that uses $O(n\sqrt{n})$ storage and that can report the hop-distance between any two nodes $s, t \in P$, up to an additive error of 2, in $O(\sqrt{n})$ time.*

▶ Remark. The theorem above is stated for n constant-complexity c-oriented polygons. However, it also holds in a more geneal setting, namely for a collection $\mathcal{P}$ of polygons with n edges in total. We can then find a separator of size $O(\sqrt{n})$ and parts A, B such that the number of polygons in A and B is at most $\frac{2|\mathcal{P}|}{3}$. Alternatively, we can guarantee that the total number of edges of the polygons in A (and similarly for B) is at most $\frac{2n}{3}$. Finally, the theorem also works in a weighted setting.

String graphs. A *string graph* is the intersection graph of a set V of curves in the plane [14] – no conditions are put on the curves and, in particular, any two curves in V can intersect arbitrarily many times. (But note that there is still at most one edge between the corresponding nodes in $\mathcal{G}^\times[V]$.) It is known that for any set U of connected regions in the plane, there is a set V of strings such that $\mathcal{G}^\times[U]$ and $\mathcal{G}^\times[V]$ are isomorphic [23]. Thus, string graphs are the most general type of intersection graphs of connected regions in the plane.

Matoušek [25] proved that any string graph with n nodes and m edges has a (node-based) separator of size $O(\sqrt{m} \log m)$.[6] Using this result we can obtain a star-based separator of sublinear size for a string graph $\mathcal{G}^\times[V]$ using the following simple two-stage process.

- *Stage 1:* As long as there is a node $v \in V$ of degree at least $n^{1/3}/\log^{2/3} n$, remove star(v) from $\mathcal{G}^\times[V]$ and add it to the separator. This puts at most $O(n^{2/3} \log^{2/3} n)$ stars into the separator.
- *Stage 2:* Construct a node-based separator on the remaining string graph, using Matoušeks method [25] and put these nodes as singletons into the separator. Since the maximum degree after Stage 1 is $O(n^{1/3}/\log^{2/3} n)$, the remaining graph has $O(n^{4/3}/\log^{2/3} n)$ edges.

[6] A paper by Lee [23] claims that a separator of size $O(\sqrt{m})$ exists for string graphs, but Bonnet et al. [29] note that there is an error in this paper. It is not yet known if the proof can be repaired.

Hence, in Stage 2 we put $O\left(\sqrt{\frac{n^{4/3}}{\log^{2/3} n}} \cdot \log\left(\frac{n^{4/3}}{\log^{2/3} n}\right)\right) = O\left(n^{\frac{2}{3}} \log^{\frac{2}{3}} n\right)$ stars into the separator.

By using the star-based separator exactly as in Section 2, this yields the following.

▶ **Proposition 14.** *Let V be a set of n strings in the plane. Then the string graph $\mathcal{G}^\times[V]$ has a star-based separator of size $O(n^{2/3} \log^{2/3} n)$. Moreover, there is an almost exact distance oracle for $\mathcal{G}^\times[V]$ that uses $O(n^{5/3} \log^{2/3} n)$ storage and that can report the hop-distance between any two nodes $s, t \in V$, up to an additive error of 2, in $O(n^{2/3} \log^{2/3} n)$ time.*

Recall that the distance oracle of Aronov, De Berg and Theocharous [12] for geodesic disks in the plane has $O(n^{7/4+\varepsilon})$ storage and $O(n^{3/4+\varepsilon})$ query time. Thus the distance oracle in Proposition 14 is more general (as it can handle any string graph), has better storage, and better query time. The only downside is that the additive error in our distance oracle is at most 2, while for their oracle it is at most 1.

▶ Remark. One may wonder if any graph, and not just any string graph, admits a sublinear star-based separator, but this is not the case. For example, 3-regular expanders do not admit sublinear node-based separators [13] and working with star-based separators instead of node-based separators does not help in constant-degree graphs.

4 Concluding remarks

Motivated by the fact that intersection graphs of non-fat objects may not admit sublinear node-based or clique-based separators, we introduced a *biclique-based* and *star-based* separators. We proved that the intersection graph of any c-colored set of pseudo-segments has a star-based separator of size $O(\sqrt{n})$, and extended the result to c-oriented polygons. We also presented a straightforward algorithm to compute a star-based separator of size $O(n^{2/3} \log^{2/3} n)$ for any string graph. These results lead to almost exact distance oracles with subquadratic storage and sublinear query times. To the best of our knowledge, such distance oracles did not yet exist – not even for intersection graphs of axis-parallel line segments.

Our work raises several questions. Can we improve the size of star-based separators for string graphs from $O(n^{2/3} \log^{2/3} n)$ to $O(\sqrt{n})$? If not, can we perhaps do so for c-colored sets of strings, or for arbitrary sets of line segments? It is also interesting to explore other applications of biclique-based separators, besides distance oracles, and to see if the bounds we obtained for distance oracles can be improved. While clique-based separators have been used to design subexponential algorithms for problems such as q-Coloring [12] and Dominating Set [11], it is unlikely that our biclique-based separator will yield new results for these problems. This is due to existing $2^{\Omega(n)}$ conditional lower bound (under ETH) for q-Coloring and Dominating Set on 2-dir and segment intersection graphs, respectively [5]. Instead, problems whose main difficulty lies in finding (hop-)distances – computing the diameter in subquadratic time [8] is an example – would be interesting to consider.

Recent work on the (weighted) Independent Set problem on restricted graph classes has exploited properties that can be interpreted through the lens of star-based separators [16, §1.4]. It is known that for every constant t, the family of graphs that does not contain an induced path of length t admits balanced separators that consist of the neighborhoods of $t-1$ vertices (cf. [17, Thm. 1.2]). Hence, such graphs have star-based separators of constant size. This property has been used to develop quasi-polynomial-time approximation schemes for weighted Independent Set on P_t-free graphs [10, 17], as well as exact subexponential-time algorithms for unweighted Independent Set on P_t-free graphs [4]. Can our star-based separators of size $O(\sqrt{n})$ also be used to obtain new algorithms for restricted input families?

References

1 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.

2 Srinivasa Rao Arikati, Danny Z. Chen, L. Paul Chew, Gautam Das, Michiel H. M. Smid, and Christos D. Zaroliagis. Planar spanners and approximate shortest path queries among obstacles in the plane. In *Proc. 4th Annual European Symposium on Algorithms (ESA 1996)*, pages 514–528, 1996. doi:10.1007/3-540-61680-2_79.

3 Boris Aronov, Mark de Berg, and Leonidas Theocharous. A clique-based separator for intersection graphs of geodesic disks in $\mathbb{R}^2$. In *Proc. 40th International Symposium on Computational Geometry (SoCG)*, pages 9:1–9:15, 2024. doi:10.4230/LIPIcs.SoCG.2024.9.

4 Gábor Bacsó, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Zsolt Tuza, and Erik Jan van Leeuwen. Subexponential-time algorithms for maximum independent set in P_t-free and broom-free graphs. *Algorithmica*, 81(2):421–438, 2019. doi:10.1007/S00453-018-0479-5.

5 Édouard Bonnet and Pawel Rzazewski. Optimality program in segment and string graphs. *Algorithmica*, 81(7):3047–3073, 2019. doi:10.1007/s00453-019-00568-7.

6 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.

7 Hsien-Chih Chang, Jie Gao, and Hung Le. Computing diameter+1 in truly subquadratic time for unit-disk graphs. *CoRR*, abs/2401.12881, 2024. doi:10.48550/arXiv.2401.12881.

8 Hsien-Chih Chang, Jie Gao, and Hung Le. Computing diameter+2 in truly-subquadratic time for unit-disk graphs. In *Proc. 40th International Symposium on Computational Geometry (SoCG)*, pages 38:1–38:14, 2024. doi:10.4230/LIPICS.SoCG.2024.38.

9 Panagiotis Charalampopoulos, Pawel Gawrychowski, Yaowei Long, Shay Mozes, Seth Pettie, Oren Weimann, and Christian Wulff-Nilsen. Almost optimal exact distance oracles for planar graphs. *J. ACM*, 70(2):12:1–12:50, 2023. doi:10.1145/3580474.

10 Maria Chudnovsky, Marcin Pilipczuk, Michal Pilipczuk, and Stéphan Thomassé. Quasi-polynomial time approximation schemes for the maximum weight independent set problem in H-free graphs. In *Proc. 31st ACM-SIAM Symposium on Discrete Algorithms (SODA 2020)*, pages 2260–2278, 2020. doi:10.1137/1.9781611975994.139.

11 Mark de Berg, Hans L. Bodlaender, Sándor Kisfaludi-Bak, Dániel Marx, and Tom C. van der Zanden. A framework for Exponential-Time-Hypothesis-tight algorithms and lower bounds in geometric intersection graphs. *SIAM J. Comput.*, 49:1291–1331, 2020. doi:10.1137/20M1320870.

12 Mark de Berg, Sándor Kisfaludi-Bak, Morteza Monemizadeh, and Leonidas Theocharous. Clique-based separators for geometric intersection graphs. *Algorithmica*, 85(6):1652–1678, 2023. doi:10.1007/S00453-022-01041-8.

13 Zdeněk Dvořák and Sergey Norin. Strongly sublinear separators and polynomial expansion. *SIAM J. Discrete Math.*, 30(2):1095–1101, 2016. doi:10.1137/15M1017569.

14 Gideon Ehrlich, Shimon Even, and Robert Endre Tarjan. Intersection graphs of curves in the plane. *J. Comb. Theory B*, 21(1):8–20, 1976. doi:10.1016/0095-8956(76)90022-8.

15 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.

16 Peter Gartland. *Quasi-Polynomial Time Techniques for Independent Set and Beyond in Hereditary Graph Classes*. PhD thesis, University of California, Santa Barbara, USA, 2023. URL: https://www.escholarship.org/uc/item/0kk6d2jv.

17 Peter Gartland, Daniel Lokshtanov, Tomás Masarík, Marcin Pilipczuk, Michal Pilipczuk, and Pawel Rzazewski. Maximum weight independent set in graphs with no long claws in quasi-polynomial time. In *Proc. 56th ACM Symposium on Theory of Computing (STOC 2024)*, pages 683–691, 2024. doi:10.1145/3618260.3649791.

18 I.Ben-Arroyo Hartman, Ilan Newman, and Ran Ziv. On grid intersection graphs. *Discrete Mathematics*, 87(1):41–52, 1991. doi:10.1016/0012-365X(91)90069-E.

19 Petr Hliněný. The maximal clique and colourability of curve contact graphs. *Discrete Applied Mathematics*, 81(1):59–68, 1998. doi:10.1016/S0166-218X(97)00075-9.

20 Jan Kratochvíl. A special planar satisfiability problem and a consequence of its NP-completeness. *Discrete Applied Mathematics*, 52(3):233–252, 1994. doi:10.1016/0166-218X(94)90143-0.

21 Jan Kratochvíl and Jaroslav Nešetřil. Independent set and clique problems in intersection-defined classes of graphs. *Commentationes Mathematicae Universitatis Carolinae*, 031(1):85–93, 1990. URL: http://eudml.org/doc/17810.

22 Hung Le and Christian Wulff-Nilsen. Optimal approximate distance oracle for planar graphs. In *Proc. 62nd IEEE Symposium on Foundations of Computer Science (FOCS 2021)*, pages 363–374. IEEE, 2021. doi:10.1109/FOCS52979.2021.00044.

23 James R. Lee. Separators in region intersection graphs. In *Proc. 8th Innovations in Theoretical Computer Science Conference (ITCS 2017)*, pages 1:1–1:8, 2017. doi:10.4230/LIPIcs.ITCS.2017.1.

24 Richard J. Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979. doi:10.1137/0136016.

25 Jirí Matoušek. Near-optimal separators in string graphs. *Comb. Probab. Comput.*, 23(1):135–139, 2014. doi:10.1017/S0963548313000400.

26 Christian Sommer. Shortest-path queries in static networks. *ACM Comput. Surv.*, 46(4):45:1–45:31, 2014. doi:10.1145/2530531.

27 Mikkel Thorup and Uri Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, 2005. doi:10.1145/1044731.1044732.

28 Jie Xue and Meirav Zehavi. Parameterized algorithms on geometric intersection graphs. *Computer Science Review*, 58:100796, 2025. doi:10.1016/j.cosrev.2025.100796.

29 Édouard Bonnet, Jędrzej Hodor, Tuukka Korhonen, and Tomáš Masařík. Treewidth is polynomial in maximum degree on weakly sparse graphs excluding a planar induced minor. *CoRR*, 2024. arXiv:2312.07962.

Optimal Online Bipartite Matching in Degree-2 Graphs

Amey Bhangale
University of California, Riverside, CA, USA

Arghya Chakraborty
Tata Institute of Fundamental Research, Mumbai, India

Prahladh Harsha
Tata Institute of Fundamental Research, Mumbai, India

Abstract

Online bipartite matching is a classical problem in online algorithms and we know that both the deterministic fractional and randomized integral online matchings achieve the same competitive ratio of $1 - \frac{1}{e}$. In this work, we study classes of graphs where the online degree is restricted to 2. As expected, one can achieve a competitive ratio of better than $1 - \frac{1}{e}$ in both the deterministic fractional and randomized integral cases, but surprisingly, these ratios are not the same. It was already known that for fractional matching, a 0.75 competitive ratio algorithm is optimal. We show that the folklore Half-Half algorithm achieves a competitive ratio of $\eta \approx 0.717772\ldots$ and more surprisingly, show that this is optimal by giving a matching lower-bound. This yields a separation between the two problems: deterministic fractional and randomized integral, showing that it is impossible to obtain a perfect rounding scheme.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Online Algorithm, Bipartite matching

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.13

Funding Research of the second and third authors supported by the Department of Atomic Energy, Government of India, under Project Identification No. RTI4001.
Amey Bhangale: Supported by Hellman Fellowship award & NSF CAREER award 2440882.
Arghya Chakraborty: Supported in part by Prof. Mrinal Kumar's SERB grant CRG/2023/006433.
Prahladh Harsha: Supported in part by the Google India Research Award.

1 Introduction

The problem of Bipartite Matching involves a bipartite graph $G = (L, R, E)$ with vertices $L \cup R$ and edges $E \subseteq L \times R$. The objective is to find the largest sized matching in G - in other words, the largest subset $M \subseteq E$ such that no two edges in M share a common vertex. This problem has been well studied [25, 4] and polynomial time algorithms are known.

Online Bipartite Matching was first introduced by Karp, Vazirani, and Vazirani [24]. In the online version of the problem, the set L is known in advance, but the vertices in R arrive one by one. Each time a vertex $j \in R$ arrives (along with its edges), we must make an irreversible decision on whether to match j with one of its neighboring vertices $i \in N(j) \subseteq L$. In a fractional version, we can assign fractional weights to the edges satisfying the matching constraints. The competitive ratio of an online matching algorithm is simply the approximation ratio achieved by the algorithm, that is, the worst-case ratio between the cost of the matching found by the algorithm and the cost of an optimal matching. Online Bipartite Matching has found immense applications in real-world problems. AdWords, organ matching, and online dating are some of the few areas with direct applications.

© Amey Bhangale, Arghya Chakraborty, and Prahladh Harsha;
licensed under Creative Commons License CC-BY 4.0

The seminal work of Karp, Vazirani, and Vazirani showed that if an unweighted graph has an optimal matching of size n then no online algorithm can guarantee a matching of size larger than $(1 - \frac{1}{e}) \cdot n$, in expectation. They also gave an algorithm, called RANKING, that attains the same competitive ratio of $(1 - \frac{1}{e})$ on unweighted graphs.

Many variants of this problem have also been studied. The survey by Mehta [27] provides an excellent overview of the problem and its variants. Among the various generalizations studied, there is the edge arrival model [20], where the edges arrive in an online fashion instead of the vertices. Recently, there have been breakthroughs [16, 30, 21, 5] in the setting where the edges are weighted (one has to make additional assumptions here, as in this case, no competitive algorithm exists without an additional assumption). Other variants, like stochastic weights [23, 15], ad words [28, 22, 13], display ads [19, 10, 17], online sub-modular welfare maximization [26, 9] have been studied. Most of these edge-weighted settings require either a stochastic input [23, 22] or a free disposal model [18, 16, 30, 5].

For all these matching-related problems, one may consider the polytope of matchings. Randomized algorithms naturally give rise to a fractional matching in this polytope. Hence, the problem of fractional matching is of extreme importance and has also been well studied. The WATER-LEVEL Algorithm [10] is a well-studied deterministic fractional matching algorithm, attaining a competitive ratio of $(1 - \frac{1}{e})$, and this is optimal. Almost all problems related to matching have this feature, which is that the optimal competitive ratio for randomized integral algorithms is the same as the optimal competitive ratio for fractional matching algorithms. Indeed, randomized integral algorithms inherently produce a probability distribution over matchings, and this probability distribution may be viewed as a fractional matching.

However, given that the competitive ratio is the same for all these related problems, it is interesting to ask if the fractional and the randomized integral matchings are inherently the same problems. Hence, people have tried to obtain integral matchings as a randomized rounding of fractional matchings [8]. In fact, for the edge weighted case, all the known algorithms [16, 30, 21, 5], first produce a fractional matching and then use (randomized) Online Correlated Selection (OCS) algorithm to perform the task of online rounding. Therefore, it is interesting to ask whether the fractional and randomized integral matching problems are the same for other variants of matching.

Bounded degree graphs. All the candidate graphs that prove the lower bound of $(1 - \frac{1}{e})$ competitive ratio for randomized integral or fractional matching have the feature that they contain very high-degree vertices. It is natural to ask whether one can construct low-degree graphs to obtain the same lower bounds. There have been some works on this [2, 1, 11, 12, 29, 6, 3]. In this paper, we study graphs where the online vertices have a degree at most 2. We show that, indeed, algorithms can perform better in this particular setting. More specifically, we prove the following theorem.

▶ **Theorem 1** (Integral Matching). *For any graph, G, with n online vertices and degree of online vertices at most 2, there exists an online randomized integral matching algorithm with competitive ratio η, and no online algorithm can attain a competitive ratio better than η,*

$$\textit{where } \eta := 1 - \sum_{i=1}^{\infty} \frac{1}{2^{2^i+i-1}} \approx 0.717772.$$

This is interesting, in contrast to the following known result:

▶ **Theorem 2** (Fractional Matching). *For any graph, G, with n online vertices and degree of online vertices at most 2, the Water-Level Algorithm (c.f, Algorithm 2) attains a competitive ratio of 0.75 and no online algorithm can obtain a better competitive ratio.*

The analysis of the fractional matching problem has been done in [6], but for the sake of completeness, we provide an easy proof in Section 5.

It is intriguing that the fractional and the randomized integral matchings seem to be different problems for this class of graphs, as the optimal competitive ratio differs for the two different settings.

We use the Primal-Dual analysis used in prior works [7, 16, 5] on online bipartite matching (described in detail in the following section) for our algorithm. Our main technical contribution is to set the primal-dual variables in the algorithm's analysis in a non-trivial manner, allowing us to prove the optimality of our algorithm.

1.1 Brief overview

In Section 2, we formally define some of the notations that we will use throughout the paper, the models that we look at, and the online primal-dual framework which is a commonly used tool in the analysis of online bipartite matching algorithms. Then in Section 4.1, we explicitly construct a graph, and then use Yao's principle to prove a lower bound for our problem. Finally, in Section 4.2 we use the online primal-dual framework to prove the tightness of the above bound, by analyzing a simple algorithm.

2 Preliminaries

Throughout this paper, we shall denote by $G(d_1, d_2)$ the set of graphs that have degree at most d_1 for the offline vertices and degree at most d_2 for the online vertices. Note that a bipartite graph on n online vertices and n offline vertices can be any graph from $G(n, n)$. We shall focus on $G(n, 2)$ graphs where the online vertices may have a degree at most 2.

We shall use L to refer to the set of offline vertices and R to refer to the set of online vertices. To make things easier to read, we shall use i, i_1, i_2 to refer to vertices in L (offline vertices) and j to refer to vertices in R (online vertices). We shall mostly consider an online vertex j, arriving with exactly two neighbors i_1 and i_2, and this is without loss of generality as given by the following lemma.

▶ **Lemma 3.** *Online Bipartite Matching on graphs in $G(n, 2)$ is equivalent to graphs with degree of online vertices exactly 2. In other words, a c-competitive randomized algorithm for one implies a c-competitive randomized algorithm for the other.*

Proof. Since $G(n, 2)$ includes graphs with online degrees at most 2, an online algorithm might perform better on graphs with exactly degree 2. We show that these two cases are equivalent. We only need to show that if we have a γ-competitive ratio algorithm on graphs with an online degree of exactly 2, then there is a γ-competitive ratio algorithm on graphs with an online degree of at most 2.

Fix an arbitrary algorithm A' with γ-competitive ratio on graphs with an online degree of exactly 2. From this, we will construct an algorithm A with γ-competitive ratio on $G(n, 2)$. Fix any graph $G = (L, R, E) \in G(n, 2)$. We construct a new graph $G'_m = (L', R', E')$, parameterized by an integer m, where all online vertices have exactly degree 2. This graph G'_m consists of m copies of G, where the i-th copy is denoted by $G_i = (L_i, R_i, E_i)$ and an additional vertex $\ell^\star$ to adjust the degree of every online vertex in G'_m to exactly two. More

precisely, the set of offline vertices L' is $L_1 \cup L_2 \cup \cdots \cup L_m \cup \{\ell^\star\}$ while the set of online vertices R' is $R_1 \cup R_2 \cup \cdots \cup R_m$. The edge set E' is $E_1 \cup E_2 \cup \cdots \cup E_m \cup \tilde{E}$ where the edges in $\tilde{E}$ are defined next.

The arrival sequence of the online vertices in G'_m is as follows: all the m copies of the first vertex in R, followed by all the m copies of the second vertex in R, and so on. Recall, each of the copies $G_i = (L_i, R_i, E_i)$ are isomorphic to the original graph $G = (L, R, E)$. The graph G'_m has an additional set $\tilde{E}$ of the edges as follows: For every $v \in R_1 \cup R_2 \cup \cdots \cup R_m$ with degree 1, we add an edge $(v, \ell^\star)$ to the graph G'_m.

Now, we run A' on G'_m for the first m online vertices, followed by the next m online vertices, and so on. The matching produced by A' can be viewed as m different matchings on G, with an additional vertex $\ell^\star \in L$. However, at most one of those matchings may contain $\ell^\star$ as part of the matching. The remaining $m-1$ matchings must not contain $\ell^\star$ and can be directly viewed as a matching on G. The algorithm A will pick one of these m matchings uniformly at random. If A happens to pick the matching with $\ell^\star$ we give up and do not out put any matching. Note that A can be generated from A' in an online fashion, by selecting one of the m matchings and simulating it. If this matching contains $\ell^\star$, we stop at that point and otherwise we output a feasible matching.

Now, let us bound the competitive ratio of A, assuming that the competitive ratio of A' is γ. Let us assume that G has an optimal matching of size n', implying that G'_m will have an optimal matching of size at least $m \cdot n'$. Hence A', with a competitive ratio of γ, should give us at least $\gamma \cdot m \cdot n'$ size, in expectation. However, in this matching produced, we might have $\ell^\star$, which we cannot use in algorithm A. By our construction, we have m collections of matchings of which one contains $\ell^\star$ and the remaining $m-1$ of them are used by A. Now, if we leave aside the particular matching that contains $\ell^\star$, we would still have a matching of size at least $\frac{\gamma \cdot m \cdot n' - n'}{m}$ in expectation, as produced by A'. The optimal matching in this set is of size at most n'. Hence the expected competitive ratio of A is at least $\frac{\gamma \cdot m \cdot n' - n'}{m \cdot n'} = \frac{\gamma \cdot m - 1}{m}$, which converges to γ as m increases. Hence, by choosing a large m, this conversion ensures that A attains the same competitive ratio as A' on G. ◀

2.1 Primal and Dual programs for bipartite matching

At this point, we would like to explain the primal and dual programs for online bipartite matching. We shall use $x_{i,j}$ to denote whether i is matched to j or not. So, we would want $x_{i,j}$ to be either 0 or 1, and the size of the matching produced would be the sum of $x_{i,j}$, which we would like to maximize. To ensure that it is a matching, we add the constraints that for each vertex, there must be at most one edge incident on the vertex.

We shall also use $x_i := \sum_{j \in R} x_{i,j}$ to denote whether vertex i is matched or not. We shall also relax this problem as follows :

Maximize $\sum_{i \in L} \sum_{j \in R} x_{i,j} = \sum_{i \in L} x_i$
subject to :

$$\forall i \in L \sum_{j \in \mathrm{Nbd}(i)} x_{i,j} \leq 1$$

$$\forall j \in R \sum_{i \in \mathrm{Nbd}(j)} x_{i,j} \leq 1$$

$$x_{i,j} \geq 0$$

Minimize $\sum_{i \in L} \alpha_i + \sum_{j \in R} \beta_j$
subject to :

$$\forall (i,j) \in E, \alpha_i + \beta_j \geq 1$$

$$\forall i \in L, j \in R, \alpha_i \geq 0 \text{ and } \beta_j \geq 0$$

We have relaxed the $x_{i,j}$'s from being $\{0,1\}$ valued to instead allow any fraction : $x_{i,j} \in [0,1]$. This means that the optimal value of the primal objective can only be larger now (since the graph is bipartite, the optimal value actually is the same), which means that the matching size obtained (the competitive ratio) will only be lower, compared to the primal optimal. Also note that if we are doing a fractional or a randomized matching, we would want $x_{i,j}$ to represent the fraction that i is matched to j or the probability of i being matched to j, ensuring that the expected size of matching is $\sum_{i \in L, j \in R} x_{i,j}$ and the change/increment in the primal value because of the vertex j is $\sum_{i \in L} x_{i,j} = \sum_{i \in L} \Delta x_i$.

The dual program represents the vertex cover problem, but we shall not use this fact in any way. Note the dual constraint $\alpha_i + \beta_j \geq 1$. If a dual solution is not feasible, meaning that the solution does not satisfy $\alpha_i + \beta_j \geq 1$ for all $(i,j) \in E$ but instead satisfies $\alpha_i + \beta_j \geq \gamma$ for some $\gamma < 1$, then we shall call this solution γ-feasible.

In the analysis of our online algorithm, we will keep updating the primal and dual solutions. We shall use ΔP and ΔD to denote the increase in the primal and dual objectives, respectively. Similarly, we shall use Δx_i and $\Delta \alpha_i$ to refer to the increments in the corresponding variables.

The main idea that we use in our algorithms is an online primal-dual method, similar to the one used in prior work [14, 16]. We always maintain feasible primal and approximately feasible dual solutions. Whenever primal increases by some amount, we increase the dual by the same amount, so the primal and dual always have the same value. The primal will be feasible, so we would obtain an optimal solution if the dual were also feasible. However, we will be unable to maintain a feasible dual solution; instead, we will maintain an approximate feasible solution that satisfies the dual constraints with a slack of γ, in other words, be γ-feasible.

More specifically, we will always maintain the following throughout :

- **Approximate Dual Feasibility:** For any $i \in L$ and $j \in R$, $\alpha_i + \beta_j \geq \gamma$
- **Reverse Weak Duality:** The objective value of the primal (P) and the objective value of the dual (D) satisfy that $P \geq D$.

This is important because of the following claim.

▷ Claim 4. If the primal solution is feasible and has value P, and if the dual solution has objective value at most P, while being γ-feasible, then the primal value P is at least γ fraction of the primal optimal.

Proof. Let M be the primal optimal. By strong duality, it is also the value of the dual optimal. Since the primal is feasible, $P \leq M$. Let the dual solution have value $D \leq P$. Since the dual solution is γ feasible, multiplying all the dual variables by $\frac{1}{\gamma}$ gives a feasible dual solution with value $\frac{D}{\gamma} \geq M$ because it is feasible.

Hence $D \leq P \leq M \leq \frac{D}{\gamma} \implies \gamma \cdot M \leq D \leq P \implies \frac{P}{M} \geq \gamma$. ◁

3 Techniques

While the algorithm that we suggest is extremely simple, the interesting part is the analysis. We use the online primal-dual method to prove that the algorithm is optimal. The primal variables are easy to update, given the algorithm. The tricky part is to update the dual variables: α_i and β_j. Indeed, there are other intuitive candidate algorithms for this problem, which we tried but were unable to analyse using the primal-dual approach.

On arrival of any vertex, $j \in R$, we try to increase the α_i's as much as possible and β_j as little as possible while maintaining γ-competitive ratio. The key observation, in Lemma 9, is that given the primal variable of a vertex $i \in L$, one can determine a lower bound for the dual variable α_i. Given this observation, it is sufficient to show that as the probability of i being matched approaches 1, the dual variable α_i approaches γ.

4 Randomized Integral Matching

We start with a lower-bound construction first.

4.1 Lower Bound

Recall that $\eta := 1 - \sum_{i=1}^{\infty} \frac{1}{2^{2^i+i-1}}$. The following theorem shows that no online algorithm can achieve a competitive ratio better than η.

▶ **Theorem 5.** *For Online Integral Bipartite Matching, no online algorithm (deterministic or randomized) can attain a competitive ratio better than η.*

Proof. We will use Yao's principle to prove this. We shall first construct one particular graph G and then look at the uniform distribution over all graphs that are isomorphic to the described graph.

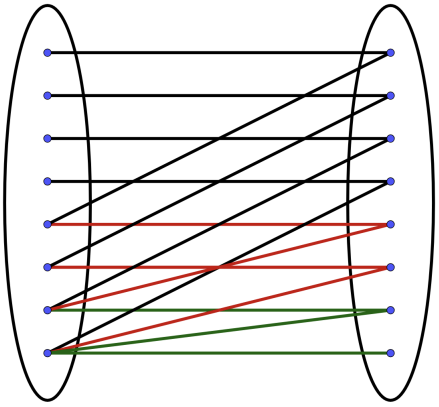


Figure 1 The graph for the lower bound, with 8 online and offline vertices with online degree bounded by 2. The black edges correspond to online vertices in the first phase, the red ones correspond to the second phase and the green ones correspond to the third phase.

This graph will have $n = 2^k$ online and 2^k offline vertices for some integer k. We will describe the graph in phases. There will be $k = \log n$ phases in total. The online i-th vertex will always have an edge to the offline i-th vertex. Since the online vertices will have degree 2, it suffices to explain the other neighbor of the online vertex i. Figure 1 depicts this graph for $k = 3$, with $2^k = 8$ online and offline vertices.

This is where we will use the phases to describe the neighbors. In the first phase, there will be $\frac{n}{2}$ online vertices: 1 through $\frac{n}{2}$ and the i-th vertex would have the other edge to offline vertex $\frac{n}{2} + i$. In the second phase, there will be $\frac{n}{4}$ online vertices: $\frac{n}{2} + 1$ through $n - \frac{n}{4}$ and the $\frac{n}{2} + i$-th vertex would have the other edge to offline vertex $n - \frac{n}{4} + i$. In the j-th phase, there will be $\frac{n}{2^j}$ online vertices: $n - \frac{n}{2^{j-1}} + 1$ through $n - \frac{n}{2^j}$ and the $n - \frac{n}{2^{j-1}} + i$-th vertex would have the other edge to offline vertex $n - \frac{n}{2^j} + i$.

Notice that in the j-th phase, when an online vertex arrives, its neighbors' degree changes from $j-1$ to j.

Now using this graph, G, let us construct a distribution over graphs. We shall consider a set of all graphs that can be obtained from G, by a permutation of the offline vertices, and then take a uniform distribution over these graphs. The idea is to use Yao's principle on this distribution and study the competitive ratio of deterministic algorithms. We shall compute the probability of offline vertices being matched, based on the phases. We shall consider any candidate deterministic algorithm for this.

Consider any offline vertex, u. If there is an online vertex, say v, in the j-th phase, with u as a neighbor of j, then we shall say that u "appeared" in the j-th phase. Notice that if u appeared in the j-phase, then u must have appeared in the phases $1, 2, \ldots, j-1$ too. For every offline vertex, u, there will be a unique j such that u appeared in the j-th phase and then did not appear in any of the phases $j+1, j+2, \ldots$ again.

Let us first look at offline vertices that appear in the first phase and then never again. The probability that such a vertex is matched is $\frac{1}{2}$, and there are $\frac{n}{2}$ such vertices.

Next, for vertices that appear in the second phase and never after that, we shall compute the probability that they are not matched. Let us consider one such vertex, u. For u not to be matched, it has to be necessarily unmatched when it appeared in the first phase. This would happen with probability $\frac{1}{2}$. Also, when u appears in the second phase, the online vertex, say v, will have another neighbor, say i_1. If i_1 happened to be matched earlier, in phase 1, then u will get matched in the second phase. Therefore, for u to be unmatched i_1 must be unmatched and then on the arrival of v the probability that the deterministic algorithm will leave u unmatched will be $\frac{1}{2}$. Hence the probability that u is unmatched is $\frac{1}{8}$.

For a vertex u that appears in the third phase and never thereafter, the probability that u is unmatched is $\frac{1}{2} \cdot \frac{1}{8} \cdot \frac{1}{8}$ because this is the case when u was unmatched even after phase two and i_1, the competitor of u in phase three was also unmatched after phase two and in this case the probability of u being unmatched is $\frac{1}{2}$.

Similarly, for a vertex that appears in the i-th phase and then never thereafter, the probability that such a vertex is unmatched is $\frac{1}{2^{2^i-1}}$. Also, there are $\frac{n}{2^i}$ such vertices that appear in the i-th phase and then never thereafter.

Hence, the expected number of vertices unmatched equals

$$\sum_{i=1}^{\infty} \frac{n}{2^{2^i+i-1}}.$$

Therefore, the competitive ratio will be

$$1 - \sum_{i=1}^{\infty} \frac{1}{2^{2^i+i-1}} = \eta \approx 0.717772.$$ ◀

4.2 Upper Bound

In this section, we will show that the folklore HALF-HALF algorithm attains a competitive ratio of η. We begin by recalling the HALF-HALF algorithm, which can be described as follows. When an online vertex v comes, if it has only one unmatched neighbor, then match v with the unmatched neighbor. If both the neighbors of v are unmatched, then pick one uniformly at random and match it with v. The formal description of the algorithm is given in Algorithm 1.

Algorithm 1 Half-Half Algorithm.

```
procedure A(j, S)                ▷ Online vertex j with neighborhood S ⊆ L, |S| ≤ 2
    if ∀i ∈ S, i is already matched then
        Do nothing.
    else
        if There exists only one i ∈ S, such that i is unmatched then
            Match i to j.
        else
            We have S = {i1, i2} such that i1 and i2 are both unmatched
            Match j with i1 and i2 with probability half each.
        end if
    end if
end procedure
```

4.2.1 Analysis of the half-half algorithm

▶ **Theorem 6.** *The half-half algorithm attains a competitive ratio of η, where:*

$$\eta := 1 - \sum_{i=1}^{\infty} \frac{1}{2^{2^i+i-1}} = 1 - \frac{1}{4} - \frac{1}{32} - \frac{1}{2^{10}} - \frac{1}{2^{19}} - \cdots \approx 0.717772.$$

Throughout the rest of the section, we are going to prove this theorem.

The plan is again to use the online Primal-Dual analysis. We will show that on the arrival of $j \in R$, with neighbors $i_1, i_2 \in L$, we can update the primal such that the expected size of the matching produced is at least the value of the primal. Simultaneously we will set up β_j and increment $\alpha_{i_1}, \alpha_{i_2}$ so that the dual conditions are η-feasible. We shall initialize by setting all the α_i's to 0, and β_j will start off as 0 before we do the increments for the dual variables. We shall describe how to update the dual variables later. First, we will describe the updates to the primal variables.

Updating the Primal variables. According to the algorithm, we can change the primal variables according to the following rule. We always maintain the primal variables to take values of the form $1 - \frac{1}{2^p}$ where $p \in \{0, 1, 2, 3, \ldots\}$. If an online vertex j comes with neighbors $\{i_1, i_2\}$, and the old values of x_{i_1} and x_{i_2} are as follows:

$$x_{i_1} = 1 - \frac{1}{2^{p_1}}, \qquad x_{i_2} = 1 - \frac{1}{2^{p_2}},$$

then the updated values would be

$$x_{i_1} = x_{i_2} = 1 - \frac{1}{2^{p_1+p_2+1}}.$$

With this, the change in the primal value would be

$$\Delta P = 2\left(1 - \frac{1}{2^{p_1+p_2+1}}\right) - \left(1 - \frac{1}{2^{p_1}}\right) - \left(1 - \frac{1}{2^{p_2}}\right).$$

▷ Claim 7. Changing the primal in this way ensures that primal variables are always of the form $1 - \frac{1}{2^p}$, and that the expected increment of matching size is at least the change in primal.

Proof. First of all, it is clear from these updates that the primal variables are always of the form $1 - \frac{1}{2^p}$. Had the events of i_1 and i_2 being unmatched been independent events, we could have said

$$\mathbb{P}[i_1 \text{ is unmatched after } j] = \mathbb{P}[i_2 \text{ is unmatched after } j]$$
$$= \frac{1}{2} \cdot \mathbb{P}[i_1 \text{ and } i_2 \text{ were unmatched before } j] = \frac{1}{2^{p_1+p_2+1}}.$$

However, if the events are not independent, then there is only a negative correlation, i.e., given that i_1 is unmatched, the probability of i_2 being unmatched can only decrease. The proof is as follows. Conditioned on i_1 being unmatched, every online vertex, with i_1 as its neighbor, must have been matched to the offline vertex other than i_1. Let us call this set of vertices S. Now, when these offline vertices from S, after being matched, were neighbors of some online vertex, say v, the online vertex v must have been matched to the other neighbor of v. We can update our S to include all such vertices that get matched in this manner. Continuing like this, the set S will consist of all vertices that are matched, conditioned on i_1 not being matched. If any online vertex had i_2 as its neighbor and a vertex from S as its other neighbor, i_2 must have been matched conditioned on i_1 not being matched. Otherwise, i_1 being matched and i_2 being matched are independent events. Hence,

$$\mathbb{P}[i_1 \text{ is unmatched after } j] = \mathbb{P}[i_2 \text{ is unmatched after } j]$$
$$= \frac{1}{2}\mathbb{P}[i_1 \text{ and } i_2 \text{ were unmatched before } j] \leq \frac{1}{2^{p_1+p_2+1}}.$$

We have already seen that the change in primal is

$$\Delta P = 2\left(1 - \frac{1}{2^{p_1+p_2+1}}\right) - \left(1 - \frac{1}{2^{p_1}}\right) - \left(1 - \frac{1}{2^{p_2}}\right) = \frac{1}{2^{p_1}} + \frac{1}{2^{p_2}} - \frac{2}{2^{p_1+p_2+1}}.$$

Now we can compare this to the expected increase in matching size,

$$\begin{aligned}
&\mathbb{E}[\text{Increment in size of matching}] \\
&\quad = \mathbb{P}[\text{Either } i_1 \text{ or } i_2 \text{ was unmatched on arrival of } j] \\
&\quad = \mathbb{P}[i_1 \text{ was unmatched on arrival of } j] + \mathbb{P}[i_2 \text{ was unmatched on arrival of } j] - \\
&\qquad\qquad \mathbb{P}[\text{Both } i_1 \text{ and } i_2 \text{ were unmatched on arrival of } j] \\
&\quad \geq \frac{1}{2^{p_1}} + \frac{1}{2^{p_2}} - \frac{2}{2^{p_1+p_2+1}}.
\end{aligned}$$

Thus,

$$\mathbb{E}[\text{Increment in size of matching}] \geq \Delta P,$$

and this proves the claim. ◁

Updating the Dual variables. Now that we have described how the primal variables are updated, we shall describe how to update the dual variables. As mentioned earlier, we will always maintain $\Delta P \geq \Delta D$. So, when online vertex j with neighbors i_1, i_2 arrives, we shall first compute ΔP and then try to set up $\alpha_{i_1} = \alpha_{i_2}$. The goal is to set up the α's as high as possible, preferably to η, and assign the remainder of ΔP to β_j if needed. This will ensure $\Delta P \geq \Delta D$. We would want $\alpha_{i_1} + \beta_j \geq \eta$ and also $\alpha_{i_2} + \beta_j \geq \eta$, while simultaneously maintaining $\beta_j \geq 0$. This would limit us from setting large α's. It will be convenient to define the following notation.

▶ **Definition 8** ($\alpha_{(k)}$). *We define the quantity $\alpha_{(k)}$ as follows: On arrival of the online vertex j with neighbors i_1 and i_2, suppose the primal variables x_{i_1} and x_{i_2} take the value $1 - \frac{1}{2^k}$ after the update, then the minimum value that the dual variables α_{i_1} and α_{i_2} are set (while maintaining η-feasibility and $\beta_j \in [0, 1]$) is the quantity $\alpha_{(k)}$.*

What we will show now is that it will always be possible for us to set the dual variables in order to maintain η-approximate dual feasibility.

We shall first show that for $1 \leq i \leq 3$, we can set up $\alpha_{(i)}$ to maintain approximate dual feasibility. After that, we will prove a general statement showing that we can assign appropriate dual variables for all x_i.

Setting $\alpha_{(1)}$. Let us first consider the case when the two neighbors i_1, i_2 had not appeared before. In this case, the half-half algorithm will set $x_{i_1} = x_{i_2} = \frac{1}{2}$. Observe that this is the only case when the x_i will be set to $\frac{1}{2}$, and hence the previous values of x_{i_1}, x_{i_2} is 0. So, $\Delta P = 1$. Let us set $\alpha_{i_1} = \alpha_{i_2} = \alpha_{(1)} = 1 - \eta$ in this case, and β_j to $2\eta - 1$. This will ensure that $\Delta D = 1 = \Delta P$ (recall, α_{i_1} and α_{i_2} were set to 0 at the beginning). Additionally, the dual constraints for the edges $\{i_1, j\}, \{i_2, j\}$ are η-satisfied.

Setting $\alpha_{(2)}$. Let the neighbors of j be i_1 and i_2 with $x_{i_1} = 1 - \frac{1}{2}$ and $x_{i_2} = 0$. Then the updated $x_{i_1} = x_{i_2} = 1 - \frac{1}{4}$ and notice that, up to symmetry, this is the only way for the primal variables to attain $x_i = 1 - \frac{1}{4}$. In this case, $\Delta P = 1$. Updating $\alpha_{i_1} = \alpha_{i_2} = \alpha_{(2)} = 2 - 2\eta$ means that the increment in $\alpha_{i_1} + \alpha_{i_2}$ is $(\alpha_{(2)} - \alpha_{(1)}) + (\alpha_{(2)} - 0) = ((2 - 2\eta) - (1 - \eta)) + ((2 - 2\eta) - 0)$ because for i_1, with $x_{i_1} = 0$, we had $\alpha_{i_1} = 1 - \eta$ and for i_2 we had $\alpha_{i_2} = 0$ before the arrival of j. Hence the total increment in $\alpha_{i_1} + \alpha_{i_2}$ is $3 - 3\eta$. Therefore, assigning β_j to $\Delta P - \Delta\alpha_{i_1} - \Delta\alpha_{i_2} = 3\eta - 2$ ensures that β_j is positive and the increment in primal is equal to the increment in the dual. Also, for the edges, the dual constraints are η-satisfied since $(2 - 2\eta) + (3\eta - 2) = \eta$.

Setting $\alpha_{(3)}$. There are two cases in this case.

1. In the first case, let the neighbors of j be i_1 and i_2 with $x_{i_1} = x_{i_2} = 1 - \frac{1}{2}$ before the arrival of j. According the the change in primal values, both x_{i_1} and x_{i_2} would be set to $1 - \frac{1}{8}$ after the update. Therefore, $\Delta P = 2(1 - \frac{1}{8}) - \frac{1}{2} - \frac{1}{2} = \frac{3}{4}$. Also, α_{i_1} and α_{i_2} used to be $\alpha_{(1)} = 1 - \eta$ before j arrived. If we update α_{i_1} and α_{i_2} to $\alpha_{(3)}$, then we can set β_j to $\Delta P - (2\alpha_{(3)} - 2(1 - \eta))$ in order to ensure that the change in dual is at most the change in primal. However, after the update, the edge corresponding to $\{i_1, j\}$ must satisfy approximate dual feasibility. Hence, we must have $\beta_j + \alpha_{(3)} \geq \eta$, implying $\frac{3}{4} - \alpha_{(3)} + 2 - 2\eta \geq \eta$. Therefore $\alpha_{(3)} \leq \frac{11}{4} - 3\eta$.
2. In the second case, if the two neighbors i_1 and i_2 had $x_{i_1} = 1 - \frac{1}{4}$ and $x_{i_2} = 0$ (i.e. i_2 not having appeared before) even then both x_{i_1} and x_{i_2} would be set to $1 - \frac{1}{8}$ after the update. It can be checked that in this case, $\Delta P = 1$ and hence we can set $\alpha_{i_1} = \alpha_{i_2} = \eta$, while keeping $\beta_j = 0$.

Considering both these cases, we can ensure that $\alpha_{i_1}, \alpha_{i_2}$ will certainly be set to at least $\frac{11}{4} - 3\eta$ and hence $\alpha_{(3)} = \frac{11}{4} - 3\eta$.

Setting $\alpha_{(k)}$ for $k \geq 4$. As we have seen till now, we can set up the α variables so that approximate dual feasibility is satisfied.

The next lemma gives a strategy for the remaining cases to set the dual variables so that the Dual is η-feasible.

▶ **Lemma 9.** *For every $k \geq 4$, there is a fixed $\alpha_{(k)}$ such that whenever the primal variables are set to $1 - \frac{1}{2^k}$, one can set the value of the corresponding dual variables α such that $\alpha \geq \alpha_{(k)}$ while ensuring $\Delta D \leq \Delta P$, $\beta_j \in [0,1]$ and the η-approximate dual feasibility. More specifically, the algorithm sets $\alpha_{(k)}$ to be η whenever k is not of the form $2^m - 1$ (for some integer m), otherwise*

$$\alpha_{(2^m-1)} \geq \sum_{i=1}^{m-1} 2^{m-1-i} \left(\frac{1}{2^{2^i-2}} - \frac{1}{2^{2(2^i-2)+2}} \right) + 2^{m-1} - (2^m - 1)\eta.$$

Before we prove this lemma, let's first see why this is enough to prove the main theorem.

Proof of Theorem 6. It is easy to see that proving this lemma completes the proof. This is because, after the arrival of j, i_1 and i_2 are more likely to be matched and hence x_{i_1} and x_{i_2} increase. We have seen that x_i may only take values of the form $1 - \frac{1}{2^k}$ for some integer k. For $k = 1$ to 3, we have explicitly checked that we can always satisfy reverse weak duality and η-approximate dual feasibility. For any larger value of k, the Lemma 9 tells us that we will always be able to set up the dual variables while satisfying reverse weak duality and η-approximate deal feasibility. Hence, we shall always be able to update the dual variables in this manner, for all k and therefore the Half-half Algorithm would be η-competitive. ◀

We now prove Lemma 9.

Proof of Lemma 9. The proof will be by induction on k. Let us assume that $\alpha_{(k)}$ follows the lemma up to some $k < T$.

We will need to check the base cases for $k = 4, 5, 6, 7$ after which the proof follows by induction. The calculations are exactly like that for $k = 1, 2, 3$. We provide a table of values here for each k. This table describes how the dual variables are set for that particular k. Each row describes a different initial setting of x_{i_1} and x_{i_2} that can lead to $x_{i_1} = x_{i_2} = 1 - \frac{1}{2^k}$. The reader may check that these updates satisfy $\Delta P \geq \Delta D$ and η-approximate dual feasibility and satisfy Lemma 9.

For $k = 4$,

x_{i_1}	x_{i_2}	ΔP	Old α_{i_1}	Old α_{i_2}	New $\alpha_{i_1} = \alpha_{i_2}$	β_j
0	$1 - \frac{1}{8}$	1	0	$\frac{11}{4} - 3\eta$	η	0
$\frac{1}{2}$	$1 - \frac{1}{4}$	$\frac{1}{2} + \frac{1}{4} - \frac{1}{8}$	$1 - \eta$	$2 - 2\eta$	η	0

For $k = 5$,

x_{i_1}	x_{i_2}	ΔP	Old α_{i_1}	Old α_{i_2}	New $\alpha_{i_1} = \alpha_{i_2}$	β_j
0	$1 - \frac{1}{16}$	1	0	η	η	0
$\frac{1}{2}$	$1 - \frac{1}{8}$	$\frac{1}{2} + \frac{1}{8} - \frac{1}{16}$	$1 - \eta$	$\frac{11}{4} - 3\eta$	η	0
$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4} + \frac{1}{4} - \frac{1}{16}$	$2 - 2\eta$	$2 - 2\eta$	η	0

For $k = 6$,

x_{i_1}	x_{i_2}	ΔP	Old α_{i_1}	Old α_{i_2}	New $\alpha_{i_1} = \alpha_{i_2}$	β_j
0	$1 - \frac{1}{32}$	1	0	η	η	0
$\frac{1}{2}$	$1 - \frac{1}{16}$	$\frac{1}{2} + \frac{1}{16} - \frac{1}{32}$	$1 - \eta$	η	η	0
$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{4} + \frac{1}{8} \frac{1}{32}$	$2 - 2\eta$	$\frac{11}{4} - 3\eta$	η	0

For $k = 7$,

x_{i_1}	x_{i_2}	ΔP	Old α_{i_1}	Old α_{i_2}	New $\alpha_{i_1} = \alpha_{i_2}$	β_j
0	$1 - \frac{1}{64}$	1	0	η	η	0
$\frac{1}{2}$	$1 - \frac{1}{32}$	$\frac{1}{2} + \frac{1}{32} - \frac{1}{64}$	$1 - \eta$	η	η	0
$\frac{1}{4}$	$\frac{1}{16}$	$\frac{1}{4} + \frac{1}{16} - \frac{1}{64}$	$2 - 2\eta$	η	η	0
$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8} + \frac{1}{8} - \frac{1}{64}$	$\frac{11}{4} - 3\eta$	$\frac{11}{4} - 3\eta$	$\frac{367}{64} - 7\eta$	$8\eta - \frac{367}{64}$

For T, we can have three cases: (1) T is of the form $2^m - 1$ for some m. (2) T is of the form $2^{m_1} + 2^{m_2} - 1$, or (3) T is not of these forms.

Case 3. We note that

$$\alpha_{(T)} = \min_{k \in \left\{0,1,2,\cdots,\frac{T-1}{2}\right\}} \left[\left(\frac{1}{2^k} + \frac{1}{2^{T-1-k}} - \frac{1}{2^{T-1}} \right) + \alpha_{(k)} + \alpha_{(T-1-k)} - \eta \right]$$

can be achieved by putting the entire ΔP into α without increasing the β. Now, considering the case we are in, we know that either $\alpha_{(k)}$ or $\alpha_{(T-1-k)}$ is η (inductive hypothesis). Since the other α already had value more than $\eta - (1 - x_i)$ (this can be seen from the fact that $\alpha_{(T)}$ converges to η as T tends to ∞ and the total value added to $\alpha_{(T)}$, at any point, is at most the increment in x_i) and observe that $\Delta P \geq (1 - x_i)$. Therefore, the alpha can be increased to η.

Case 1. In this case $T = 2^m - 1$ for some m.

$$\alpha_{(2^m-1)} = \min_{k \in \{0,1,2,\cdots,2^{m-1}-1\}} \left(\frac{1}{2^k} + \frac{1}{2^{2^m-k-2}} - \frac{1}{2^{2^m-2}} \right) + \alpha_{(k)} + \alpha_{(2^m-k-2)} - \eta$$

Note that the minimum occurs when $k = 2^{m-1} - 1$ (on top of this, we need to show that $\Delta P - \Delta\alpha \geq 0$, so that β can be set to a value in $[0, 1]$). In this case,

$$\begin{aligned}
\alpha_{(2^m-1)} &= \left(\frac{1}{2^{2^{m-1}-1}} + \frac{1}{2^{2^m-2^{m-1}+1-2}} - \frac{1}{2^{2^m-2}} \right) + \alpha_{(2^{m-1}-1)} + \alpha_{(2^m-2^{m-1}+1-2)} - \eta \\
&= \sum_{i=1}^{m-1} 2^{m-1-i} \left(\frac{1}{2^{2^i-2}} - \frac{1}{2^{2(2^i-2)+2}} \right) + 2^{m-1} - (2^m - 1)\eta.
\end{aligned}$$

as required. Now, we need to show that the following quantity is non-negative.

$$\begin{aligned}
\Delta P - \Delta\alpha &= \left(\frac{1}{2^{2^{m-1}-1}} + \frac{1}{2^{2^{m-1}-1}} - \frac{1}{2^{2^m-2}} \right) - 2 \cdot \left(\alpha_{(2^m-1)} - \alpha_{(2^{(m-1)}-1)} \right). \\
&= \left(\frac{1}{2^{2^{m-1}-1}} + \frac{1}{2^{2^{m-1}-1}} - \frac{1}{2^{2^m-2}} \right) - 2 \cdot \left(\alpha_{(2^m-1)} - \alpha_{(2^{(m-1)}-1)} \right). \\
&= \left(\frac{1}{2^{2^{m-1}-2}} - \frac{1}{2^{2^m-2}} \right) - 2 \cdot \left(\alpha_{(2^m-1)} - \alpha_{(2^{(m-1)}-1)} \right).
\end{aligned}$$

We have the following claim.

▷ Claim 10. The quantity $\left(\frac{1}{2^{2^{m-1}-2}} - \frac{1}{2^{2^m-2}} \right) - 2 \cdot \left(\alpha_{(2^m-1)} - \alpha_{(2^{(m-1)}-1)} \right)$ is non-negative for all values of $m \geq 1$.

Proof. We first simplify the last term.

$$\begin{aligned}
&\left(\alpha_{(2^m-1)} - \alpha_{(2^{(m-1)}-1)}\right) \\
&= \left(\sum_{i=1}^{m-1} 2^{m-1-i}\left(\frac{1}{2^{2^i-2}} - \frac{1}{2^{2(2^i-2)+2}}\right) + 2^{m-1} - (2^m-1)\eta\right) \\
&\quad - \left(\sum_{i=1}^{m-2} 2^{m-2-i}\left(\frac{1}{2^{2^i-2}} - \frac{1}{2^{2(2^i-2)+2}}\right) + 2^{m-2} - (2^{m-1}-1)\eta.\right) \\
&= \frac{1}{2}\cdot\left(\frac{1}{2^{2^{m-1}-2}} - \frac{1}{2^{2(2^{m-1}-2)+2}}\right) + (2^{m-1} - 2^{m-2}) - (2^m - 2^{m-1})\eta \\
&\quad + \sum_{i=1}^{m-1}(2^{m-1-i} - 2^{m-2-i})\left(\frac{1}{2^{2^i-2}} - \frac{1}{2^{2(2^i-2)+2}}\right) \\
&= \frac{1}{2}\cdot\left(\frac{1}{2^{2^{m-1}-2}} - \frac{1}{2^{2(2^{m-1}-2)+2}}\right) + 2^{m-2} - 2^{m-1}\left(1 - \sum_{i=1}^{\infty}\frac{1}{2^{2^i+i-1}}\right) \\
&\quad + \sum_{i=1}^{m-1} 2^{m-2-i}\left(\frac{1}{2^{2^i-2}} - \frac{1}{2^{2(2^i-2)+2}}\right) \\
&= \frac{1}{2}\cdot\left(\frac{1}{2^{2^{m-1}-2}} - \frac{1}{2^{2(2^{m-1}-2)+2}}\right) - 2^{m-2} + 2^{m-2}\sum_{i=1}^{\infty}\frac{2}{2^{2^i+i-1}} \\
&\quad + 2^{m-2}\cdot\sum_{i=1}^{m-1}\frac{1}{2^i}\cdot\left(\frac{1}{2^{2^i-2}} - \frac{1}{2^{2(2^i-2)+2}}\right).
\end{aligned}$$

Now,

$$\begin{aligned}
2\left(\alpha_{(2^m-1)} - \alpha_{(2^{(m-1)}-1)}\right) &= \left(\frac{1}{2^{2^{m-1}-2}} - \frac{1}{2^{2(2^{m-1}-2)+2}}\right) - 2^{m-1} + 2^{m-1}\sum_{i=1}^{\infty}\frac{2}{2^{2^i+i-1}} \\
&\quad + 2^{m-1}\cdot\sum_{i=1}^{m-1}\frac{1}{2^i}\cdot\left(\frac{1}{2^{2^i-2}} - \frac{1}{2^{2(2^i-2)+2}}\right) \\
&= \left(\frac{1}{2^{2^{m-1}-2}} - \frac{1}{2^{2^m-2}}\right) - 2^{m-1} + 2^{m-1}\sum_{i=1}^{\infty}\frac{2}{2^{2^i+i-1}} \\
&\quad + 2^{m-1}\cdot\sum_{i=1}^{m-1}\frac{1}{2^i}\cdot\left(\frac{1}{2^{2^i-2}} - \frac{1}{2^{2(2^i-2)+2}}\right).
\end{aligned}$$

Therefore,

$$\begin{aligned}
\Delta P - \Delta\alpha &= 2^{m-1} - 2^{m-1}\sum_{i=1}^{\infty}\frac{2}{2^{2^i+i-1}} - 2^{m-1}\cdot\sum_{i=1}^{m-1}\frac{1}{2^i}\cdot\left(\frac{1}{2^{2^i-2}} - \frac{1}{2^{2(2^i-2)+2}}\right) \\
&= 2^{m-1}\left(1 - \sum_{i=1}^{\infty}\frac{2}{2^{2^i+i-1}} - \sum_{i=1}^{m-1}\frac{1}{2^i}\cdot\left(\frac{1}{2^{2^i-2}} - \frac{1}{2^{2(2^i-2)+2}}\right)\right) \\
&= 2^{m-1}\left(1 - \sum_{i=1}^{\infty}\frac{2}{2^{2^i+i-1}} - \sum_{i=1}^{m-1}\left(\frac{2}{2^{2^i+i-1}} - \frac{1}{2^{(2^{i+1}+i-2)}}\right)\right).
\end{aligned}$$

Now, we simplify the last summation as follows

$$\begin{aligned}\sum_{i=1}^{m-1}\left(\frac{2}{2^{2^i+i-1}}-\frac{1}{2^{(2^{i+1}+i-2)}}\right) &= \sum_{i=1}^{m-1}\left(\frac{2}{2^{2^i+i-1}}-\frac{1}{2^{(2^{(i+1)}+(i+1)-3)}}\right)\\ &= \sum_{i=1}^{m-1}\left(\frac{2}{2^{2^i+i-1}}-\frac{4}{2^{(2^{(i+1)}+(i+1)-1)}}\right)\\ &= \frac{2}{2^2}-\sum_{i=2}^{m-1}\frac{2}{2^{2^i+i-1}}-\frac{4}{2^{(2^m+m-1)}}\end{aligned}$$

Hence,

$$\begin{aligned}\Delta P-\Delta\alpha &= 2^{m-1}\left(1-\sum_{i=1}^{\infty}\frac{2}{2^{2^i+i-1}}-\sum_{i=1}^{m-1}\left(\frac{2}{2^{2^i+i-1}}-\frac{1}{2^{(2^{i+1}+i-2)}}\right)\right)\\ &= 2^{m-1}\left(1-\sum_{i=1}^{\infty}\frac{2}{2^{2^i+i-1}}-\frac{2}{2^2}+\sum_{i=2}^{m-1}\frac{2}{2^{2^i+i-1}}+\frac{4}{2^{(2^m+m-1)}}\right)\\ &= 2^{m-1}\left(1-\frac{2}{2^2}-\sum_{i=m}^{\infty}\frac{2}{2^{2^i+i-1}}-\frac{2}{2^2}+\frac{4}{2^{(2^m+m-1)}}\right)\\ &= 2^{m-1}\left(\frac{4}{2^{(2^m+m-1)}}-\sum_{i=m}^{\infty}\frac{2}{2^{2^i+i-1}}\right)\geq 0.\end{aligned}$$

◁

Given the claim, we can see that $\Delta P-\Delta\alpha\geq 0$, allowing us to set $\beta_j\geq 0$. ◀

Case 2. In the case when T is of the form $2^{m_1}+2^{m_2}-1$ (where $m_1>m_2$), we will try to find out $\alpha_{(T)}$ using $\alpha_{(2^{m_1}-1)}$ and $\alpha_{(2^{m_2}-1)}$. For all other instances in which we end up with $\alpha_{(T)}$, at least one of the offline vertices will already have α set to η so like case 1, we are in good shape. Hence we just need to consider the case when one offline vertex, say i_1, had x_{i_1} at $1-\frac{1}{2^{2^{m_1}-1}}$ and the other vertex, i_2, had x_{i_2} at $1-\frac{1}{2^{2^{m_2}-1}}$.

First of all, we note that

$$\Delta P=\frac{1}{2^{2^{m_1}-1}}+\frac{1}{2^{2^{m_2}-1}}-\frac{2}{2^{2^{m_1}+2^{m_2}-1}}.$$

The plan is to put all the ΔP into both the α values and confirm that indeed we can make both the α values equal to η, without incrementing the β.

This can be achieved iff ΔP is at least $2\eta-\alpha_{(2^{m_1}-1)}-\alpha_{(2^{m_2}-1)}$. The following claim precisely shows this.

▷ Claim 11. $\Delta P\geq 2\eta-\alpha_{(2^{m_1}-1)}-\alpha_{(2^{m_2}-1)}$.

Proof. We show that $\Delta P-2\eta+\alpha_{(2^{m_1}-1)}+\alpha_{(2^{m_2}-1)}$ is non-negative as follows.

$$\begin{aligned}&\Delta P-2\eta+\alpha_{(2^{m_1}-1)}+\alpha_{(2^{m_2}-1)}\\ &= \frac{1}{2^{2^{m_1}-1}}+\frac{1}{2^{2^{m_2}-1}}-\frac{2}{2^{2^{m_1}+2^{m_2}-1}}+\sum_{i=1}^{m_1-1}2^{m_1-1-i}\left(\frac{1}{2^{2^i-2}}-\frac{1}{2^{2(2^i-2)+2}}\right)+\\ &\qquad\sum_{i=1}^{m_2-1}2^{m_2-1-i}\left(\frac{1}{2^{2^i-2}}-\frac{1}{2^{2(2^i-2)+2}}\right)+2^{m_1-1}+2^{m_2-1}-(2^{m_1}+2^{m_2})\eta\end{aligned}$$

$$
\begin{aligned}
&= \frac{1}{2^{2^{m_1}-1}} + \frac{1}{2^{2^{m_2}-1}} - \frac{2}{2^{2^{m_1}+2^{m_2}-1}} + 2^{m_1-1} + 2^{m_2-1} - (2^{m_1}+2^{m_2})\eta + \\
&\qquad 2^{m_1-1}\left(\frac{1}{2} - \sum_{i=2}^{m_1-1}\frac{2}{2^{2^i+i-1}} - \frac{4}{2^{(2^{m_1}+m_1-1)}}\right) + \\
&\qquad 2^{m_2-1}\left(\frac{1}{2} - \sum_{i=2}^{m_1-1}\frac{2}{2^{2^i+i-1}} - \frac{4}{2^{(2^{m_1}+m_1-1)}}\right) \\
&= -\frac{1}{2^{2^{m_1}-1}} - \frac{1}{2^{2^{m_2}-1}} - \frac{2}{2^{2^{m_1}+2^{m_2}-1}} + \\
&\qquad 2^{m_1-1}\left(1 - 2 + 2\sum_{i=1}^{\infty}\frac{1}{2^{2^i+i-1}} + \frac{1}{2} - \sum_{i=2}^{m_1-1}\frac{2}{2^{2^i+i-1}}\right) + \\
&\qquad 2^{m_2-1}\left(1 - 2 + 2\sum_{i=1}^{\infty}\frac{1}{2^{2^i+i-1}} + \frac{1}{2} - \sum_{i=2}^{m_2-1}\frac{2}{2^{2^i+i-1}}\right) \\
&= -\frac{1}{2^{2^{m_1}-1}} - \frac{1}{2^{2^{m_2}-1}} - \frac{2}{2^{2^{m_1}+2^{m_2}-1}} + 2^{m_1-1}\left(\sum_{i=m_1}^{\infty}\frac{2}{2^{2^i+i-1}}\right) + \\
&\qquad 2^{m_2-1}\left(\sum_{i=m_2}^{\infty}\frac{2}{2^{2^i+i-1}}\right) \\
&\geq 0
\end{aligned}
$$

◁

5 Fractional Matching

In this section, we prove a lower bound on the competitive ratio of any algorithm for fractional matching and then give an online algorithm that achieves the best attainable competitive ratio.

5.1 Lower Bound

▶ **Theorem 12.** *No online algorithm can attain a competitive ratio better than* 0.75 *for online fractional bipartite matching.*

Proof. Consider a graph with two offline vertices i_1 and i_2. The first online vertex will have edges to both i_1 and i_2 while the second online vertex will have an edge to only one of the vertices. The optimal matching is of size 2 but one can see that no online algorithm can attain a competitive ratio better than 0.75 using this graph. ◀

5.2 Upper Bound

For the fractional matching, we shall consider the water-level algorithm which was an optimal algorithm for the case without the degree bound.

Algorithm 2 Water-Level Algorithm.

procedure $A(j, S)$ ▷ Online vertex j with neighborhood $S \subseteq L$, $|S| \leq 2$
  Find ℓ such that $\sum_{i \in S} \max\{0, \ell - x_i\}=1$
  if $\ell \leq 1$ **then**
    Vertex j can be completely matched.
    For all $i \in S$ with $x_i < \ell$, set $x_i = \ell$
  else
    All the neighbors of j can be completely matched, but j can only be partially matched.
    For all $i \in S$ set $x_i = 1$
  end if
end procedure

We show that the same algorithm gives the optimal algorithm for the degree 2 case.

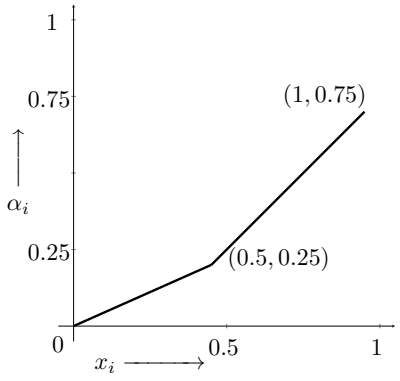
▶ **Theorem 13.** *Water-level algorithm attains a competitive ratio of* 0.75 *when the vertices have a degree at most* 2*.*

Proof. We shall use the primal-dual analysis for this. The main idea is to maintain the value of the fractional matching produced by water level and simultaneously maintain dual variables.

We already know how the primal variables are updated by water level, given the algorithm. We will now explain how the dual variables will be updated to maintain reverse weak duality and weak dual feasibility.

Let us define $x_i := \sum_{j \in R} x_{i,j}$. The variable x_i denotes how much the vertex i is matched. We will maintain the α_i's such that, given x_i, we will know exactly what α_i is. Thus, we will know how much α_i increases by considering how much x_i was incremented by water level. Given the increments in α_i's, we will know what value to set β_j because we will maintain that the primal and dual objectives will always be the same. So, the dual objective will be increased by the same amount as the primal, which is the amount of matching done by the water level algorithm on the arrival of online vertex j.

Hence, first, we will describe how to maintain the α_i's.



For the first half fraction of the matching, for any vertex $i \in L$, if a fractional matching of size p is done between i and $j \in R$, then we shall increment α_i by $p/2$ and β_j by $p/2$. Whereas, if u is already matched more than 0.5, then for a matching of size x, we shall increment α_i by x, and we shall not increment β_j at all.

Firstly, note that we never decrease any dual variables. So if some dual constraint is satisfied at a given time then it will certainly be satisfied in the future. We shall now see that whenever a new vertex $j \in R$ arrives, both the newly introduced edges (i_1, j) and (i_2, j) satisfy approximate dual feasibility. This will complete the proof.

In order to observe the approximate dual feasibility, we shall look at a few cases based on how much i_1 and i_2 were matched right before the arrival of j.

1. **i_1 and i_2 are completely unmatched.**
   Notice that in this case, the water-level algorithm will match j half to i_1 and a half to i_2, and the primal increases by 1. Let us see how the dual variables are updated. Since x_{i_1} and x_{i_2} are both at 0.5 we will have $\alpha_{i_1} = \alpha_{i_1} = 0.25$ which adds 0.5 to the dual objective. Hence, we can add 0.5 to β_j.
   Therefore $\alpha_{i_1} + \beta_j = \alpha_{i_2} + \beta_j = 0.75$
2. **i_1 is x_{i_1} fraction matched and i_2 is completely unmatched**
   We will assume that i_1 is x_{i_1} fraction matched where $x_{i_1} \neq 0$. First, we note that the water-level algorithm maintains that if $x_{i_1} \neq 0$, then we must have $x_{i_1} \geq 0.5$. In other words, if a vertex $i \in L$ starts getting matched, it will be matched at least till the half-level mark. Hence we may assume $x_{i_1} \geq 0$.
   In this case, water-level algorithm will first match x_{i_1} fraction of vertex i_2 to j and then $\frac{1-x_{i_1}}{2}$ to i_1 and $\frac{1-x_{i_1}}{2}$ to i_2. This will mean that $\frac{1+x_{i_1}}{2}$ will be the updated value of x_{i_1} and x_{i_2} after vertex j is matched fractionally.
   Hence, after the matching $\alpha_{i_1} = \alpha_{i_2} = 0.25 + \frac{x_{i_1}}{2}$ and $\beta_j = 0.25$. Therefore $\alpha_{i_1} + \beta_j = \alpha_{i_2} + \beta_j = 0.5 + \frac{x_{i_1}}{2} \geq 0.75$ since x_{i_1} was greater than $\frac{1}{2}$.
   Note that the case where i_1 is completely unmatched but i_2 is partially matched is symmetric.
3. **i_1 is x_{i_1} fraction matched and i_2 is x_{i_2} fraction matched**
   In this last case, when both i_1 and i_2 were partially matched to begin with (hence, at least half matched as stated earlier), we can see that the water-level algorithm would end up matching i_1 and i_2 to j - whatever fraction of i_1 and i_2 was still unmatched. After this, we will have $x_{i_1} = x_{i_2} = 1$ and therefore $\alpha_{i_1} = \alpha_{i_2} = 0.75$ and therefore $\alpha_{i_1} + \beta_j = \alpha_{i_2} + \beta_j = 0.75$. ◀

6 Conclusion

An interesting open problem is finding the right competitive ratios for the integral online bipartite matching for (online) degree d unweighted graphs. If we denote the competitive ratios by $\mathcal{F}(d)$, for fractional, and $\mathcal{I}(d)$, for integral, respectively, then we have shown that $\mathcal{I}(2) = 0.717772\ldots$. Given that $\mathcal{F}(2) = 0.75$, it would be interesting to find the values of $\mathcal{I}(d)$ when $d \geq 3$ and compare them with $\mathcal{F}(d)$ especially given that we know both $\mathcal{I}(d)$ and $\mathcal{F}(d)$ approach $1 - \frac{1}{e}$ as d approaches infinity. We conjecture that $\mathcal{F}(d)$ and $\mathcal{I}(d)$ will be different for all $d \geq 2$.

It is also interesting to ask the more general question: given an algorithm that computes fractional solutions for a given class of graphs, how well can we round a fractional solution into a randomized integral solution?

References

1 Susanne Albers and Sebastian Schubert. Online ad allocation in bounded-degree graphs. In Kristoffer Arnsfelt Hansen, Tracy Xiao Liu, and Azarakhsh Malekian, editors, *Proc. 18th International Conference on Web and Internet Economics (WINE)*, volume 13778 of *LNCS*, pages 60–77. Springer, 2022. doi:10.1007/978-3-031-22832-2_4.

2 Susanne Albers and Sebastian Schubert. Tight bounds for online matching in bounded-degree graphs with vertex capacities. In Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman, editors, *Proc. 30th Annual European Symp. of Algorithms (ESA)*, volume 244 of *LIPIcs*, pages 4:1–4:16. Schloss Dagstuhl, 2022. doi:10.4230/LIPICS.ESA.2022.4.

3 Yossi Azar, Ilan Reuven Cohen, and Alan Roytman. Online lower bounds via duality. In Philip N. Klein, editor, *Proc. 28th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 1038–1050, 2017. doi:10.1137/1.9781611974782.66.

4 Claude Berge. Two theorems in graph theory. *PNAS*, 43(9):842–844, 1957. doi:10.1073/pnas.43.9.842.

5 Guy Blanc and Moses Charikar. Multiway online correlated selection. In Jelani Nelson, editor, *Proc. 63rd IEEE Symp. on Foundations of Comp. Science (FOCS)*, pages 1277–1284, 2022. doi:10.1109/FOCS52979.2021.00124.

6 Niv Buchbinder, Kamal Jain, and Joseph (Seffi) Naor. Online primal-dual algorithms for maximizing ad-auctions revenue. In Lars Arge, Michael Hoffmann, and Emo Welzl, editors, *Proc. 15th Annual European Symp. of Algorithms (ESA)*, volume 4698 of *LNCS*, pages 253–264. Springer, 2007. doi:10.1007/978-3-540-75520-3_24.

7 Niv Buchbinder and Joseph (Seffi) Naor. The design of competitive online algorithms via a primal-dual approach. *Found. Trends Theor. Comput. Sci.*, 3(2-3):93–263, 2009. doi:10.1561/0400000024.

8 Niv Buchbinder, Joseph (Seffi) Naor, and David Wajc. Lossless online rounding for online bipartite matching (despite its impossibility). In Nikhil Bansal and Viswanath Nagarajan, editors, *Proc. 34th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 2030–2068, 2023. doi:10.1137/1.9781611977554.CH78.

9 Deeparnab Chakrabarty and Gagan Goel. On the approximability of budgeted allocations and improved lower bounds for submodular welfare maximization and GAP. *SIAM J. Comput.*, 39(6):2189–2211, 2010. (Preliminary version in *49th FOCS*, 2008). doi:10.1137/080735503.

10 Denis Xavier Charles, Max Chickering, Nikhil R. Devanur, Kamal Jain, and Manan Sanghi. Fast algorithms for finding matchings in lopsided bipartite graphs with applications to display ads. In David C. Parkes, Chrysanthos Dellarocas, and Moshe Tennenholtz, editors, *Proc. 11th ACM Conf. Electronic Commerce (EC)*, pages 121–128. ACM, 2010. doi:10.1145/1807342.1807362.

11 Ilan Reuven Cohen and Binghui Peng. Primal-dual schemes for online matching in bounded degree graphs. In Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman, editors, *Proc. 31st Annual European Symp. of Algorithms (ESA)*, volume 274 of *LIPIcs*, pages 35:1–35:17. Schloss Dagstuhl, 2023. doi:10.4230/LIPICS.ESA.2023.35.

12 Ilan Reuven Cohen and David Wajc. Randomized online matching in regular graphs. In Artur Czumaj, editor, *Proc. 29th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 960–979, 2018. doi:10.1137/1.9781611975031.62.

13 Nikhil R. Devanur and Thomas P. Hayes. The adwords problem: online keyword matching with budgeted bidders under random permutations. In John Chuang, Lance Fortnow, and Pearl Pu, editors, *Proc. 10th ACM Conf. on Electronic Commerce (EC)*, pages 71–78. ACM, 2009. doi:10.1145/1566374.1566384.

14 Nikhil R. Devanur, Kamal Jain, and Robert D. Kleinberg. Randomized primal-dual analysis of RANKING for online bipartite matching. In Sanjeev Khanna, editor, *Proc. 24th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 101–107, 2013. doi:10.1137/1.9781611973105.7.

15 Tomer Ezra, Michal Feldman, Nick Gravin, and Zhihao Gavin Tang. Online stochastic max-weight matching: Prophet inequality for vertex and edge arrival models. In Péter Biró, Jason D. Hartline, Michael Ostrovsky, and Ariel D. Procaccia, editors, *Proc. 21st ACM Conf. Economics and Comput. (EC)*, pages 769–787. ACM, 2020. doi:10.1145/3391403.3399513.

16 Matthew Fahrbach, Zhiyi Huang, Runzhou Tao, and Morteza Zadimoghaddam. Edge-weighted online bipartite matching. *J. ACM*, 69(6):45:1–45:35, 2022. (Preliminary version in *61st FOCS*, 2020). doi:10.1145/3556971.

17 Jon Feldman, Monika Henzinger, Nitish Korula, Vahab S. Mirrokni, and Clifford Stein. Online stochastic packing applied to display ad allocation. In Mark de Berg and Ulrich Meyer, editors, *Proc. 18th Annual European Symp. of Algorithms (ESA), Part I*, volume 6346 of *LNCS*, pages 182–194. Springer, 2010. doi:10.1007/978-3-642-15775-2_16.

18 Jon Feldman, Nitish Korula, Vahab S. Mirrokni, S. Muthukrishnan, and Martin Pál. Online ad assignment with free disposal. In Stefano Leonardi, editor, *Proc. 5th International Workshop on Internet and Network Economics (WINE)*, volume 5929 of *LNCS*, pages 374–385. Springer, 2009. doi:10.1007/978-3-642-10841-9_34.

19 Jon Feldman, Aranyak Mehta, Vahab S. Mirrokni, and S. Muthukrishnan. Online stochastic matching: Beating $1-\frac{1}{e}$. In Daniel A. Spielman, editor, *Proc. 50th IEEE Symp. on Foundations of Comp. Science (FOCS)*, pages 117–126, 2009. doi:10.1109/FOCS.2009.72.

20 Buddhima Gamlath, Michael Kapralov, Andreas Maggiori, Ola Svensson, and David Wajc. Online matching with general arrivals. In David Zuckerman, editor, *Proc. 60th IEEE Symp. on Foundations of Comp. Science (FOCS)*, pages 26–37, 2019. doi:10.1109/FOCS.2019.00011.

21 Ruiquan Gao, Zhongtian He, Zhiyi Huang, Zipei Nie, Bijun Yuan, and Yan Zhong. Improved online correlated selection. In Nisheeth Vishnoi, editor, *Proc. 62nd IEEE Symp. on Foundations of Comp. Science (FOCS)*, pages 1265–1276, 2021. doi:10.1109/FOCS52979.2021.00123.

22 Gagan Goel and Aranyak Mehta. Online budgeted matching in random input models with applications to adwords. In Shang-Hua Teng, editor, *Proc. 19th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 982–991, 2008. URL: http://dl.acm.org/citation.cfm?id=1347082.1347189.

23 Bernhard Haeupler, Vahab S. Mirrokni, and Morteza Zadimoghaddam. Online stochastic weighted matching: Improved approximation algorithms. In Ning Chen, Edith Elkind, and Elias Koutsoupias, editors, *Proc. 7th International Workshop on Internet and Network Economics (WINE)*, volume 7090 of *LNCS*, pages 170–181. Springer, 2011. doi:10.1007/978-3-642-25510-6_15.

24 Richard M. Karp, Umesh V. Vazirani, and Vijay V. Vazirani. An optimal algorithm for on-line bipartite matching. In Harriet Ortiz, editor, *Proc. 22nd ACM Symp. on Theory of Computing (STOC)*, pages 352–358, 1990. doi:10.1145/100216.100262.

25 Harold W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955. doi:10.1002/nav.3800020109.

26 Benny Lehmann, Daniel Lehmann, and Noam Nisan. Combinatorial auctions with decreasing marginal utilities. *Games Econ. Behav.*, 55(2):270–296, 2006. doi:10.1016/J.GEB.2005.02.006.

27 Aranyak Mehta. Online matching and ad allocation. *Found. Trends Theor. Comput. Sci.*, 8(4):265–368, 2013. doi:10.1561/0400000057.

28 Aranyak Mehta, Amin Saberi, Umesh V. Vazirani, and Vijay V. Vazirani. Adwords and generalized online matching. *J. ACM*, 54(5):22, 2007. (Preliminary version in *46th FOCS*, 2005). doi:10.1145/1284320.1284321.

29 Joseph (Seffi) Naor and David Wajc. Near-optimum online ad allocation for targeted advertising. *ACM Trans. Economics and Comput.*, 6(3-4):16:1–16:20, 2018. (Preliminary version in *16th EC*, 2015). doi:10.1145/3105447.

30 Yongho Shin and Hyung-Chan An. Making three out of two: Three-way online correlated selection. In Hee-Kap Ahn and Kunihiko Sadakane, editors, *Proc. 32nd International Symposium on Algorithms and Computation (ISAAC)*, volume 212 of *LIPIcs*, pages 49:1–49:17. Schloss Dagstuhl, 2021. doi:10.4230/LIPICS.ISAAC.2021.49.

On the (In)Approximability of the Monitoring Edge Geodetic Set Problem

Davide Bilò
Department of Information Engineering, Computer Science, and Mathematics, University of L'Aquila, Italy

Giordano Colli
Department of Information Engineering, Computer Science, and Mathematics, University of L'Aquila, Italy

Luca Forlizzi
Department of Information Engineering, Computer Science, and Mathematics, University of L'Aquila, Italy

Stefano Leucci
Department of Information Engineering, Computer Science, and Mathematics, University of L'Aquila, Italy

Abstract

We study the minimum *Monitoring Edge Geodetic Set* (MEG-Set) problem introduced in [Foucaud et al., CALDAM'23]: given a graph G, we say that an edge is monitored by a pair u, v of vertices if *all* shortest paths between u and v traverse e; the goal is to find a subset M of vertices of G such that each edge of G is monitored by at least one pair of vertices in M, and $|M|$ is minimized.

In this paper, we prove that all polynomial-time approximation algorithms for the minimum MEG-Set problem must have an approximation ratio of $\Omega(\log n)$, unless $\mathsf{P} = \mathsf{NP}$. To the best of our knowledge, this is the first non-constant inapproximability result known for this problem. We also strengthen the known NP-hardness of the problem on 2-apex graphs by showing that the same result holds for 1-apex graphs. This leaves open the question of determining whether the problem remains NP-hard on planar (i.e., 0-apex) graphs.

On the positive side, we design an algorithm that computes good approximate solutions for hereditary graph classes that admit efficiently computable balanced separators of truly sublinear size. This immediately yields polynomial-time approximation algorithms achieving an approximation ratio of $O(n^{\frac{1}{4}}\sqrt{\log n})$ on planar graphs, graphs with bounded genus, and k-apex graphs with $k = O(n^{\frac{1}{4}})$. On graphs with bounded treewidth, we obtain an approximation ratio of $O(\log^{3/2} n)$. This compares favorably with the best-known approximation algorithm for general graphs, which achieves an approximation ratio of $O(\sqrt{n \log n})$ via a simple reduction to the Set Cover problem.

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

Keywords and phrases Monitoring Edge Geodetic Set, Inapproximability, Approximation Algorithms

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.14

Funding This work was partially supported by the MONET (Mutual visibility On NETworks) project funded by the University of L'Aquila.

1 Introduction

Consider a communication network that might suffer link failures. Such a network can be modeled as an undirected and connected graph G, where vertices represent hosts and edges correspond to links between hosts. In order to quickly detect failure events, the designer wishes to equip the hosts of the network with additional *probes*: probes can communicate with one another and they can monitor the distance between their corresponding nodes in the network (e.g., via a traceroute mechanism). Whenever the distance between a pair of

© Davide Bilò, Giordano Colli, Luca Forlizzi, and Stefano Leucci;
licensed under Creative Commons License CC-BY 4.0

vertices increases, this indicates that some communication link is inoperative, and an alarm can be raised. Observe that, in the above scenario, it may be possible for two probes to be connected by two (or more) edge-disjoint shortest paths, in which case they will not be able to detect an edge failure.

This motivates the following problem: given a network G, find the smallest possible set M of vertices to equip with probes while ensuring that each edge e of G has at least one pair of vertices $u, v \in M$ for which all shortest paths between u and v in G traverse e. The above problem is known as the minimum *Monitoring Edge Geodetic Set* (MEG-Set) problem and was introduced in [9], where the authors focus on providing upper and lower bounds on the size of minimum MEG-Sets for both general graphs and special graph classes (trees, cycles, unicyclic graphs, complete graphs, grids, hypercubes, ...).

Further bounds on the size of MEG-Sets have been given for the Cartesian and strong products of two graphs [12], for other graph products including join and direct products [21], as a function of the graph's girth and chromatic number [6], for ladder, butterfly, circulant and Benes networks, for convex polytopes [20], and for radix triangular mesh networks and Sierpiński graphs [17]. Moreover, the minimum MEG-Set problem was proven to be NP-hard on general graphs [12], NP-hard on 3-degenerate 2-apex graphs [8], and APX-hard on 4-degenerate graphs [7]. If the Exponential Time Hypothesis [13] holds, then the problem cannot be solved in subexponential time on 3-degenerate graphs [7]. In the same paper, the authors also provide an exact polynomial-time algorithm for interval graphs and two FPT algorithms: one for general graphs parameterized by the sum of cliquewidth and diameter, and the other for chordal graphs parameterized by treewidth. Polynomial-time algorithms for computing optimal MEG-Sets are also known for distance-hereditary graphs, P_4-sparse graphs, bipartite permutation graphs, and strongly chordal graphs [10].

As far as approximation algorithms are concerned, it has been observed [2, 1, 7] that a simple reduction to the Set Cover problem yields a polynomial-time algorithm returning solutions of size $O(k^2 \log n)$, where k is the size of a MEG-Set of minimum cardinality, thus achieving an approximation ratio of $O(\min\{k \log n, n/k\}) = O(\sqrt{n \log n})$.[1]

1.1 Our results

In this paper, we investigate the approximability and inapproximability of the minimum MEG-Set problem.

We prove that, if $\mathsf{P} \neq \mathsf{NP}$, the problem admits no polynomial-time approximation algorithm with an approximation factor of $o(\log n)$. Moreover, the problem admits no $(\alpha \log n)$-approximation algorithm for any constant $\alpha < \frac{1}{2}$ unless $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{O(\log \log n)})$. To the best of our knowledge, these are the first non-constant inapproximability results for the minimum MEG-Set problem.

We also extend the aforementioned $O(\sqrt{n \log n})$-approximation algorithm for minimum MEG-Set to a generalized version of the problem in which edges have non-negative *lengths*, vertices have binary *costs*, and a minimum-cost subset of vertices that monitors *a given subset* of edges needs to be selected. We use this result as a building block to design a more involved approximation algorithm for minimum MEG-Set that relies on recursively computing vertex-separators and provides good approximations for graph families that admit sparse balanced vertex-separators. More precisely, we achieve an approximation ratio of

[1] To the best of our knowledge, this algorithm was first given and analyzed in a Bachelor's thesis [2], and the result was subsequently claimed in a short communication based thereon [1]. The same algorithm, and its analysis, also appear in a later unpublished work [7] by a different group of authors.

$O(n^{\frac{1}{4}}\sqrt{\log n})$ on planar graphs, on graphs with bounded genus, and on k-apex graphs when $k = O(n^{\frac{1}{4}})$. We also obtain an approximation ratio of $O(\log^{3/2} n)$ on graphs with bounded treewidth.

Finally, we show that the problem remains NP-hard even on 1-apex graphs (while the reduction of [6] uses 2-apex graphs). It is an interesting open problem to settle the complexity status of the problem for planar graphs.

1.2 Organization of the paper

We start by giving some preliminary definitions and notation in Section 2. Our inapproximability result for general graphs is presented in Section 4, while the NP-Hardness result for 1-apex graphs is given in Section 3. Finally, Section 5 is devoted to our approximation algorithms.

2 Preliminaries and notation

Throughout the paper, we consider simple, connected, and undirected graphs unless otherwise specified. Given a graph $G = (V, E)$, we may use $V(G)$ or $E(G)$ to refer to the vertex-set V or the edge-set E, respectively. We denote the open neighborhood of a vertex $v \in V(G)$ in a graph G as $N_G(v) = \{u \in V(G) : \{u, v\} \in E(G)\}$; in the same way, the closed neighborhood of a vertex $v \in V(G)$ in G is defined as $N_G[v] = N_G(v) \cup \{v\}$. When the graph is clear from the context, we refer to the open (resp. closed) neighborhood of a vertex $v \in V(G)$ as $N(v)$ (resp. $N[v]$). We also use n to refer to $|V(G)|$. For a set $C \subseteq V$, we define $N(C) = \bigcup_{v \in C} N(u)$ and $N[C] = \bigcup_{v \in C} N[v]$.

We say that an edge $(u, v) \in E(G)$ is *monitored* by a pair of distinct vertices $x, y \in V(G)$ if (u, v) lies on *all* the *shortest* paths between u and v. Similarly, a set of edges E' is monitored by $M \subseteq V$ if each edge $e \in E'$ is monitored by at least one pair of vertices in M. A *monitoring edge-geodetic set* MEG-Set of $G = (V, E)$ is a subset of vertices that monitors E.

A walk π between two vertices u and v in an edge-weighted graph G with weight function w is an ordered list of alternating (and not necessarily distinct) vertices and edges $\langle v_0, e_1, v_1, e_2, v_2, \ldots, e_k, v_k \rangle$ with $v_0 = u$, $v_k = v$, and such that $e_i = (v_{i-1}, v_i)$ for all $i = 1, \ldots, k$. The weight $w(\pi)$ of π is $\sum_{i=1}^{k} w(e_i)$.

We conclude this section by stating two well-known structural properties of MEG-Sets.

▶ **Lemma 1** ([2, 9]). *All vertices of degree* 1 *in* G *belong to all MEG-Sets of* G.

▶ **Lemma 2** ([2, 6]). *Let* u *be a vertex of degree* 1 *in* G *and let* v *be its sole neighbor. If* $|V(G)| \geq 3$ *and* M *is a MEG-Set of* G*, then* $M \setminus \{v\}$ *is a MEG-Set of* G.

3 NP-Hardness of minimum MEG-Set on graphs with apex 1

In this section, we show a reduction from the minimum Planar Dominating Set problem on graphs with girth at least seven (which is known to be NP-Hard [22]) to the minimum MEG-Set problem on graphs with apex 1.

Let $G = (V, E)$ be a planar input graph with $|V(G)| \geq 2$ and with girth $g(G) \geq 7$. We construct an instance of MEG-Set, namely H, starting from G and augmenting it as follows: for each vertex $v \in V(G)$, we add 2 vertices v' and v'' to G, then connect v to v' and v' to v''. We denote the set containing all the vertices v' as L' and the set containing all the vertices v'' as L''. Notice that:

i) Each $v'' \in L''$ belongs to every MEG-Set of G due to Lemma 1;

ii) Each $v' \in L'$ does not belong to any optimal MEG-Set of G due to Lemma 2.

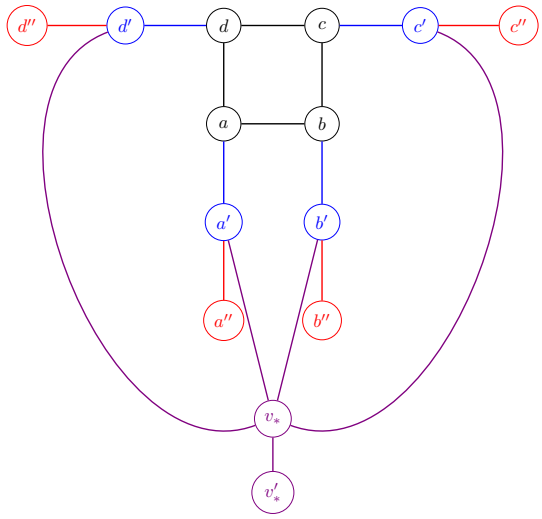


Figure 1 An example of the construction of the graph H from the input graph $G = (\{a, b, c, d\}, \{\ \{a, b\}, \{b, c\}, \{c, d\}, \{d, a\}\ \})$, which is shown in black.

The basic idea is to monitor each edge that is incident to a vertex $v \in V(G)$ that belongs to a *selected* MEG-Set in H. We denote the set of edges (v, v') as $E_{L'}$ and the set of edges (v', v'') as $E_{L''}$. Note that at this point of the construction, all edges are monitored by at least one pair of leaves in L''. To avoid this, we shortcut all the shortest paths from one vertex in L'' to another by inserting two vertices v_* and v'_* and connecting them together. Then, we connect v_* to all vertices $v' \in L'$. We denote the set containing edges (v_*, v') with $v' \in L'$ as E_*. The vertex v_* is used to create shortcuts between each shortest path from a vertex in $L' \cup L''$ to a vertex in $L' \cup L''$. Notice that, since v'_* is a leaf, it belongs to every MEG-Set of G due to Lemma 1. The vertex v'_* is used to monitor the edges in E_*. Formally, let $H = (V(G) \cup L' \cup L'' \cup \{v_*, v'_*\}, E(G) \cup E_{L'} \cup E_{L''} \cup E_* \cup \{\ \{v_*, v'_*\}\ \})$, where:

- $L' = \{v' : v \in V(G)\}$;
- $L'' = \{v'' : v \in V(G)\}$;
- $E_{L'} = \{\{v, v'\} : v \in V(G)\}$;
- $E_{L''} = \{\{v'', v'\} : v \in V(G)\}$;
- $E_* = \{\{v_*, v'\} : v \in V(G)\}$.

We prove the NP-Completeness of minimum MEG-Set on graphs with apex 1 via two lemmas. The first lemma shows that if there is a MEG-Set of the graph H having size at most $k + |V(G)| + 1$, there exists a Dominating Set of G having size at most k. Conversely, the second lemma shows that if there is a Dominating Set of the graph G having size at most k, there exists a MEG-Set of H having size $k + |V(G)| + 1$.

▶ **Lemma 3.** *Let D be a Dominating Set of G of size at most k. Then $M = D \cup L'' \cup \{v'_*\}$ is a MEG-Set of H and $|M| \leq k + |V(G)| + 1$.*

Proof. We prove that each edge $e \in E(H)$ is monitored by M. We distinguish four cases:

- Case 1: $e \in E(G)$. Let $e = (u, v)$. Since D is a Dominating Set of G, there are two subcases:
  - At least one of u and v must belong to D, therefore we assume w.l.o.g. that $u \in D$. Each path from v'' to u must either pass through the vertex v or pass through the vertex v_*. The unique shortest path including v is trivially $\langle (v'', v'), (v', v), (v, u) \rangle$ and has length 3. Each path passing through v_* must contain at least four edges. Therefore, (u, v) is monitored by $\{v'', u\}$, where both v'' and u are in M since $v'' \in L''$ and $u \in D$.

- Neither u nor v is in D. First, we observe that u and v are dominated by two distinct vertices in D; otherwise, there would be an induced triangle in G, which is not allowed due to $g(G) \geq 7$. Let these two distinct vertices be $\{x, y\} \subseteq D$. They have distance $d(x, y)$ equal to 3 since, if this were not true, the edge (u, v) would not be monitored by the pair $\{x, y\}$ due to the existence of a shortest path of length 4 that uses only edges in $E(H) \setminus E(G)$. Finally, since $g(G) \geq 7$, it can be shown that there is no alternative shortest path of length 3 from x to y.

- Case 2: $e \in E_{L''} \cup E_*$. In this case, there is *exactly* one endpoint u' of e that belongs to L'. It follows that either $e = (u'', u')$ with $u'' \in L''$ or $e = (v_*, u')$. Notice that there is a unique shortest path from v'_* to u'' including the edges $(v'_*, v_*), (v_*, u'), (u', u'')$. This implies that the edge e is monitored by $\{v'_*, u''\}$, where both v'_* and $u'' \in L''$ are in M.
- Case 3: $e \in E_{L'}$. Let $e = (u, u')$ where u is the unique endpoint of e in $V(G)$. Since D dominates all the vertices in $V(G)$, either $u \in D$, and we let $x = u$, or $u \notin D$ and there exists some vertex $x \in D \cap N(u)$. There is a single shortest path from u'' to x, and it traverses the edge (u, u').
- Case 4: $e = (v'_*, v_*)$. All shortest paths from v'_* to any other vertex in H must traverse e, and hence e is monitored, e.g., by all pairs $\{v'_*, u\}$ with $u \in D$. Note that there exists at least one vertex $u \neq v'_*$ in M since $|M| \geq 2$.

The cardinality $|M|$ is equal to the sum of the cardinality of $M \setminus V(G)$ and the cardinality of $M \cap V(G)$, with $(M \setminus V(G)) \cap (M \cap V(G)) = \emptyset$. Since $|M \cap V(G)|$ is equal to the size of the Dominating Set of G, namely k, and $M \setminus V(G) = L'' \cup \{v'_*\}$, we can conclude that:

$$|M| = |M \cap V(G)| + |M \setminus V(G)| = k + |L''| + |\{v'_*\}| = k + |V(G)| + 1.$$ ◀

▶ **Lemma 4.** *Let M be a MEG-Set of H with size at most $k + |V(G)| + 1$ (for a suitable value of k). Then, $M \cap V(G)$ is a Dominating Set of G with size at most k.*

Proof. Let $M' = M \setminus (L' \cup \{v_*\})$ be the set obtained by removing the vertex v_* and all the vertices of L' from M. Due to Lemma 2, M' is a MEG-Set of G. Moreover, since $(L' \cup \{v_*\}) \cap V(G) = \emptyset$, we can observe that $M' \cap V(G) = M \cap V(G)$. Let $D = M' \setminus (L'' \cup \{v'_*\})$ be the set of vertices in the MEG-Set M' that are in $V(G)$. We prove that if $v \in V(G)$, then either $v \in D$ or $N(v) \cap D \neq \emptyset$. Towards a contradiction, suppose that $v \notin D$ and no vertices in $N(v)$ are in D. Since M' is a MEG-Set of H, it monitors every edge in $E(G) \subseteq E(H)$. Consider an edge (u, v) with $v \in N(u)$. (u, v) must be monitored by a pair of vertices x, y not in $\{u\} \cup N(u)$. It follows that the distance between x and y is at least four, but there is a path between x and y of length four that only uses edges in $E(H) \setminus E(G)$; thus, the edge (u, v) cannot be monitored by the pair $\{x, y\}$, which is a contradiction. The above discussion shows that $M' \cap V(G)$ is a Dominating Set of G. The size of such a Dominating Set is $|M' \cap V(G)| \leq |M \cap V(G)| = |M| - |M \setminus V(G)| \leq |M| - (|L''| + 1) \leq |M| - (|V(G)| + 1) \leq k$, where we used $(L'' \cup \{v'_*\}) \subseteq M \setminus V(G)$, as ensured by Lemma 1. ◀

▶ **Theorem 5.** *The MEG-Set decision problem is* NP*-Complete on graphs with apex* 1.

Proof. For each planar graph G with girth at least seven, it is possible to construct in polynomial time a graph H with apex 1, as discussed in this section. From Lemmas 3 and 4, there exists a Dominating Set D of G such that $|D| \leq k$ if and only if there exists a MEG-Set M of H such that $|M| \leq k + |V(G)| + 1$. Since the Dominating Set problem is NP-Hard even for planar graphs with girth at least seven, this implies that the MEG-Set decision problem is NP-Hard on graphs with apex 1. ◀

4 Inapproximability of MEG-Set

We reduce from the Set Cover problem. A Set Cover instance $\mathcal{I} = \langle X, \mathcal{S} \rangle$ is described as a set of η items $X = \{x_1, \ldots, x_\eta\}$ and a collection $\mathcal{S} = \{S_1, \ldots, S_h\}$ of $h \geq 2$ distinct subsets of X, such that each subset contains at least two items and each item appears in at least two subsets.[2] The goal is that of computing a collection $\mathcal{S}^* \subseteq \mathcal{S}$ of minimum size such that $\cup_{S_i \in \mathcal{S}^*} S_i = X$.[3] It is known that, unless $\mathsf{P} = \mathsf{NP}$, all polynomial-time approximation algorithms for the Set Cover must have an approximation ratio of $(1 - o(1)) \ln \eta$ when $h = O(\text{poly}(\eta))$ [18, 4]. Moreover, unless $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{O(\log \log n)})$, the same inapproximability result holds even for $h \leq \eta$ [5].

Given an instance $\mathcal{I} = \langle X, \mathcal{S} \rangle$ of Set Cover, we build an associated bipartite graph H whose vertex set $V(H)$ is $X \cup \mathcal{S}$ and such that H contains the edge (x_i, S_j) if and only if $x_i \in S_j$. We define $N = h + \eta$.

Let k be an integer parameter, whose exact value will be chosen later, that satisfies $2 \leq k = O(\text{poly}(N))$. We construct a graph G that contains k copies $H_1, \ldots, H_k$ of H as induced subgraphs. In the following, for any $\ell = 1, \ldots, k$, we denote by $x_{i,\ell}$ and $S_{j,\ell}$ the vertices of H_ℓ corresponding to the vertices x_i and S_j of H, respectively. More precisely, we build G by starting with a graph that contains exactly the k copies $H_1, \ldots, H_k$ of H and augmenting it as follows (see Figure 2):

- For each item $x_i \in X$, we add two new vertices y_i, y'_i along with the edge (y_i, y'_i) and all the edges in $\{(x_{i,\ell}, y_i) \mid \ell = 1, \ldots, k\}$;
- We add all the edges $(y_i, y_{i'})$ for all $1 \leq i < i' \leq \eta$, so that the subgraph induced by $y_1, \ldots, y_\eta$ is complete.
- For each set $S_j \in \mathcal{S}$, we add two new vertices z_j, z'_j along with the edge (z_j, z'_j), and all the edges in $\{(S_{j,\ell}, z_j) \mid \ell = 1, \ldots, k\}$.

Observe that the number n of vertices of G satisfies $n = 2h + 2\eta + k(\eta + h) = (k+2)(\eta + h) = (k+2)N$.

Let $Y = \{y_i \mid i = 1, \ldots, \eta\}$, $Y' = \{y'_i \mid i = 1, \ldots, \eta\}$, $Z = \{z_j \mid j = 1, \ldots, h\}$, and $Z' = \{z_j \mid i = 1, \ldots, h\}$. Moreover, define L as the set of all vertices of degree 1 in G, i.e., $L = Y' \cup Z'$. By Lemma 1, the vertices in L belong to all MEG-Sets of G.

▶ **Lemma 6.** *L monitors all edges having both endvertices in $Y \cup Y' \cup Z \cup Z'$.*

Proof. Observe that all shortest paths from $y'_i \in L$ (resp. $z'_j \in L$) to any other vertex $v \in L \setminus \{y'_i\}$ (resp. $v \in L \setminus \{z'_j\}$) must traverse the sole edge incident to y'_i (resp. z'_j), namely (y'_i, y_i) (resp. (z'_j, z_j)). Since $|L| \geq 2$, such a v always exists, and all edges incident to L are monitored by L.

The only remaining edges are those with both endpoints in Y. Let $(y_i, y_{i'})$ be such an edge. Since the only shortest path between y'_i and $y'_{i'}$ in G is $\langle y'_i, y_i, y_{i'}, y'_{i'} \rangle$, the pair $\{y'_i, y'_{i'}\}$ monitors $(y_i, y_{i'})$. ◀

▶ **Lemma 7.** *Let $\mathcal{S}_1, \ldots, \mathcal{S}_k$ be k (not necessarily distinct) set covers of $\mathcal{I}$. The set $M = L \cup \{S_{j,\ell} \mid S_j \in \mathcal{S}_\ell, 1 \leq \ell \leq k\}$ is a MEG-Set of G.*

---

[2] This can be guaranteed w.l.o.g. by repeatedly reducing the instance by applying the first applicable of the following two reduction rules. Rule 1: if there exists an item x_i that is contained in a single subset S_j, then S_j belongs to all feasible solutions, and we reduce to the instance in which both S_j and x_i have been removed. Rule 2: if there exists a subset S_j that contains a single element, then (due to Rule 1) there is an optimal solution that does not contain S_j, and we reduce to the instance in which S_j has been removed. Notice that this process can only decrease the values of η and h.

[3] We assume w.l.o.g. that $\cup_{i=1}^{h} S_i = X$, i.e., that a solution exists.

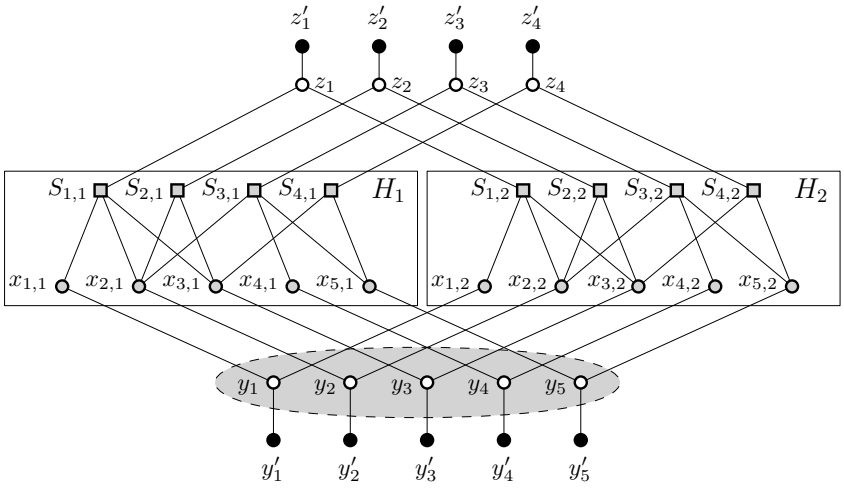


Figure 2 The graph G obtained by applying our reduction with $k = 2$ to the Set Cover instance $\mathcal{I} = \langle X, \mathcal{S} \rangle$ with $\eta = 5$, $h = 4$, $S_1 = \{x_1, x_2, x_3\}$, $S_2 = \{x_2, x_3\}$, $S_3 = \{x_2, x_4, x_5\}$, and $S_4 = \{x_3, x_5\}$. To reduce clutter, the edges of the clique induced by the vertices y_i (in the gray area) are not shown.

Proof. Since $L \subseteq M$, by Lemma 6, we only need to argue about edges with at least one endvertex in some H_ℓ, with $1 \leq \ell \leq k$. Let $S_j \in \mathcal{S}_\ell$, and consider any $x_i \in S_j$. Edge $(S_{j,\ell}, z_j)$ is monitored by $\{z'_j, S_{j,\ell}\}$. Edges $(S_{j,\ell}, x_{i,\ell})$ and $(x_{i,\ell}, y_i)$ are monitored by $\{S_{j,\ell}, y'_i\}$.

The only remaining edges with at least one endvertex in H_ℓ are those incident to vertices $S_{j,\ell}$ with $S_j \in \mathcal{S} \setminus \mathcal{S}_\ell$. Consider any such S_j, let x_i be an item in S_j, and let $S_{k'} \in \mathcal{S}_\ell$ be any set such that $x_i \in S_{k'}$ (notice that both x_i and $S_{k'}$ exist since sets are non-empty and $\mathcal{S}_\ell$ is a set cover). Edge $(S_{j,\ell}, x_{i,\ell})$ is monitored by $\{S_{k',\ell}, z'_j\}$, which also monitors $(S_{j,\ell}, z_j)$. ◀

We say that a MEG-Set M is *minimal* if, for every $v \in M$, $M \setminus \{v\}$ is not a MEG-Set. Lemma 2 ensures that any minimal MEG-Set M does not contain any of the vertices y_i, for $i = 1, \ldots, \eta$, or z_j for $j = 1, \ldots, h$. Hence, $M \setminus L$ contains only vertices in $\bigcup_{\ell=1}^{k} V(H_\ell)$. The following lemma characterizes the structure of minimal MEG-Sets of G.

▶ **Lemma 8.** *Let M be a minimal MEG-Set of G. For every $i = 1, \ldots, \eta$ and every $\ell = 1, \ldots, k$, at least one of the following conditions is true: (i) $x_{i,\ell} \in M$; or (ii) there exists an index j such that $x_i \in S_j$ and the set M contains $S_{j,\ell}$.*

Proof. Let $S(x_{i,\ell})$ be the set of all $S_{j,\ell}$ such that $x_i \in S_j$. We show that no pair $\{u, v\}$ with $u, v \in M \setminus (\{x_{i,\ell}\} \cup S(x_{i,\ell}))$ can monitor the edge $(x_{i,\ell}, y_i)$.

Notice that, since M is minimal, M does not contain any vertex $y_{i'}$, for $i' = 1, \ldots, \eta$, nor any vertex $z_{j'}$ for $j' = 1, \ldots, h$.

Let $\lambda \in \{1, \ldots, h\} \setminus \{\ell\}$ (such a λ always exists since $h \geq 2$), and notice that any path P in G that has both endvertices in $V(G) \setminus V(H_\ell)$ can always be transformed into a walk P' with as many edges as P by replacing any occurrence of a vertex from copy H_ℓ of H with the occurrence of the corresponding vertex from copy H_λ. Since P' does not contain $x_{i,\ell}$, this implies that no pair $\{u, v\}$ with $u, v \in V(G) \setminus V(H_\ell)$ can monitor $(x_{i,\ell}, y_i)$.

The above discussion allows us to restrict ourselves to the situation in which at least one vertex of $\{u, v\}$ belongs to $V(H_\ell)$. We sat that a vertex of G is an *item-vertex* if it is some $x_{i',\ell'}$ with $i' = 1, \ldots, \eta$ and $\ell' = 1, \ldots, k$. Analogously, a vertex of G is a *set-vertex* if it is some $S_{j',\ell'}$ with $j' = 1, \ldots, h$ and $\ell' = 1, \ldots, k$. We consider the following cases and, for each of them, we exhibit a shortest path $\widetilde{P}$ from u to v in G that does not traverse $(x_{i,\ell}, y_i)$:

Case 1: u is an item-vertex in H_ℓ and v is a set-vertex. In this case, $u = x_{i',\ell}$ for some $i' \neq i$, and we distinguish the following sub-cases:

- If $v = S_{j',\ell}$ and $x_{i'} \in S_{j'}$, then $\widetilde{P}$ consists of the edge $(x_{i',\ell}, S_{j',\ell})$.
- If $v = S_{j',\ell}$, $x_{i'} \notin S_{j'}$, and there exists some $S_{j''}$ such that $x_{i'} \in S_{j''}$ and $S_{j'} \cap S_{j''} \neq \emptyset$, then let $x_{i''} \in S_{j'} \cap S_{j''}$. We choose $\widetilde{P} = \langle x_{i',\ell}, S_{j'',\ell}, x_{i'',\ell}, S_{j',\ell} \rangle$.
- If $v = S_{j',\ell}$, $x_{i'} \notin S_{j'}$, and there exists no $S_{j''}$ such that $x_{i'} \in S_{j''}$ and $S_{j'} \cap S_{j''} \neq \emptyset$, then let $x_{i''} \in S_{j'}$ (notice that $x_i \notin S_{j'}$), and choose $\widetilde{P} = \langle x_{i',\ell}, y_{i'}, y_{i''}, x_{i'',\ell}, S_{j',\ell} \rangle$.
- If $v = S_{j',\ell'}$, $x_{i'} \in S_{j'}$, and $\ell' \neq \ell$, then choose $\widetilde{P} = \langle x_{i',\ell}, S_{j',\ell}, z_{j'}, S_{j',\ell'} \rangle$.
- If $v = S_{j',\ell'}$, $x_{i'} \notin S_{j'}$, and $\ell' \neq \ell$, then let $x_{i''} \in S_{j'}$ (notice that $x_i \notin S_{j'}$) and choose $\widetilde{P} = \langle x_{i',\ell}, y_{i'}, y_{i''}, x_{i'',\ell'}, S_{j',\ell'} \rangle$.

Case 2: u is an item-vertex in H_ℓ and v is an item-vertex. In this case, $u = x_{i',\ell}$ for some $i' \neq i$, and we distinguish the following sub-cases:

- If $v = x_{i'',\ell}$ with $i'' \neq i$ and there exists some set $S_{j'}$ such that $x_{i'}, x_{i''} \in S_{j'}$, then choose $\widetilde{P} = \langle x_{i',\ell}, S_{j',\ell}, x_{i'',\ell} \rangle$.
- If $v = x_{i'',\ell}$ with $i'' \neq i$ and there is no set $S_{j'}$ such that $x_{i'}, x_{i''} \in S_{j'}$, then choose $\widetilde{P} = \langle x_{i',\ell}, y_{i'}, y_{i''}, x_{i'',\ell} \rangle$.
- If $v = x_{i',\ell''}$ with $\ell'' \neq \ell$, then choose $\widetilde{P} = \langle x_{i',\ell}, y_{i'}, x_{i',\ell''} \rangle$.
- If $v = x_{i'',\ell''}$ with $i'' \neq i'$ and $\ell'' \neq \ell$, then choose $\widetilde{P} = \langle x_{i',\ell}, y_{i'}, y_{i''}, x_{i'',\ell''} \rangle$.

Case 3: u is a set-vertex in H_ℓ and v is a set-vertex. In this case, $u = S_{j',\ell}$ for some j' such that $x_{i,\ell} \notin S_{j'}$, and we distinguish the following sub-cases:

- If $v = S_{j'',\ell}$ and there exists some $x_{i'}$ such that $x_{i'} \in S_{j'} \cap S_{j''}$, then $i' \neq i$, and we pick $\widetilde{P} = \langle S_{j',\ell}, x_{i',\ell}, S_{j'',\ell} \rangle$.
- If $v = S_{j'',\ell}$, $S_{j'} \cap S_{j''} = \emptyset$, and there exists some $S_{j'''}$ such that $S_{j'} \cap S_{j'''} \neq \emptyset$ and $S_{j''} \cap S_{j'''} \neq \emptyset$, then choose any $x_{i'} \in S_{j'} \cap S_{j'''}$ and any $x_{i''} \in S_{j''} \cap S_{j'''}$. We pick $\widetilde{P} = \langle S_{j',\ell}, x_{i',\ell}, S_{j''',\ell}, x_{i'',\ell}, S_{j'',\ell} \rangle$.
- If $v = S_{j'',\ell}$, $S_{j'} \cap S_{j''} = \emptyset$, and there is no $S_{j'''}$ such that $S_{j'} \cap S_{j'''} \neq \emptyset$ and $S_{j''} \cap S_{j'''} \neq \emptyset$, then choose $x_{i'} \in S_{j'} \setminus \{x_i\}$, $x_{i''} \in S_{j''} \setminus \{x_i\}$, and pick $\widetilde{P} = \langle S_{j',\ell}, x_{i',\ell}, y_{i'}, y_{i''}, x_{i'',\ell}, S_{j'',\ell} \rangle$.
- If $v = S_{j',\ell'}$ with $\ell' \neq \ell$, then we pick $\widetilde{P} = \langle S_{j',\ell}, z_{j'}, S_{j',\ell'} \rangle$.
- If $v = S_{j'',\ell'}$ with $j'' \neq j'$ and $\ell' \neq \ell$, and there exists some item $x_{i'} \in S_{j'} \cap S_{j''}$, then we choose $\widetilde{P} = \langle S_{j',\ell}, z_{j'}, S_{j',\ell'}, x_{i',\ell'}, S_{j'',\ell'} \rangle$.
- If $v = S_{j'',\ell'}$ with $j'' \neq j'$ and $\ell' \neq \ell$, and $S_{j'} \cap S_{j''} = \emptyset$, then let $x_{i'} \in S_{j'} \setminus \{x_i\}$, $x_{i''} \in S_{j''} \setminus \{x_i\}$. We pick $\widetilde{P} = \langle S_{j',\ell}, x_{i',\ell}, y_{i'}, y_{i''}, x_{i'',\ell'}, S_{j'',\ell'} \rangle$.

Case 4: u is an item-vertex in H_ℓ and $v \in L$. In this case, $u = x_{i',\ell}$ for some $i' \neq i$, and we distinguish the following sub-cases:

- If $v = y'_{i'}$, then $\widetilde{P} = \langle x_{i',\ell}, y_{i'}, y'_{i'} \rangle$.
- If $v = y'_{i''}$ with $i'' \neq i'$, then $\widetilde{P} = \langle x_{i',\ell}, y_{i'}, y_{i''}, y'_{i''} \rangle$.
- If $v = z'_{j'}$ and $x_{i'} \in S_{j'}$, then pick $\widetilde{P} = \langle x_{i',\ell}, S_{j',\ell}, z_{j'}, z'_{j'} \rangle$.
- If $v = z'_{j'}$, $x_{i'} \notin S_{j'}$, and there exists some $S_{j''}$ for which $S_{j'} \cap S_{j''} \neq \emptyset$, then let $x_{i''} \in S_{j'} \cap S_{j''}$ and choose $\widetilde{P} = \langle x_{i',\ell}, S_{j'',\ell}, x_{i'',\ell}, S_{j',\ell}, z_{j'}, z'_{j'} \rangle$.
- If $v = z'_{j'}$, $x_{i'} \notin S_{j'}$, and there is no $S_{j''}$ for which $S_{j'} \cap S_{j''} \neq \emptyset$, then let $x_{i''} \in S_{j'} \setminus \{x_i\}$. We choose $\widetilde{P} = \langle x_{i',\ell}, y_{i'}, y_{i''}, x_{i'',\ell}, S_{j',\ell}, z_{j'}, z'_{j'} \rangle$.

Case 5: u is a set-vertex in H_ℓ and $v \in L$. In this case, $u = S_{j',\ell}$ for some j' such that $x_{i,\ell} \notin S_{j'}$, and we distinguish the following sub-cases:

- If $v = y'_{i'}$ with $x_{i'} \in S_{j'}$, then $i' \neq i$, and we choose $\widetilde{P} = \langle S_{j',\ell}, x_{i',\ell}, y_{i'}, y'_{i'} \rangle$.
- If $v = y'_{i'}$ with $x_{i'} \notin S_{j'}$, then let $x_{i''} \in S_{j'}$ and choose $\widetilde{P} = \langle S_{j',\ell}, x_{i'',\ell}, y_{i''}, y_{i'}, y'_{i'} \rangle$.
- If $v = z'_{j'}$, then $\widetilde{P} = \langle S_{j',\ell}, z_{j'}, z'_{j'} \rangle$.

- If $v = z'_{j''}$ with $j'' \neq j'$ and $S_{j'} \cap S_{j''} \neq \emptyset$, then let $x_{i'} \in S_{j'} \cap S_{j''}$ and choose $\widetilde{P} = \langle S_{j',\ell}, x_{i',\ell}, S_{j'',\ell}, z_{j''}, z'_{j''} \rangle$.
- If $v = z'_{j''}$ with $j'' \neq j'$, $S_{j'} \cap S_{j''} = \emptyset$, and there exists some $S_{j'''}$ such that $S_{j'} \cap S_{j'''} \neq \emptyset$ and $S_{j''} \cap S_{j'''} \neq \emptyset$, then let $x_{i'} \in S_{j'} \cap S_{j'''}$ and $x_{i''} \in S_{j''} \cap S_{j'''}$. We choose $\widetilde{P} = \langle S_{j',\ell}, x_{i',\ell}, S_{j''',\ell}, x_{i'',\ell}, S_{j'',\ell}, z_{j''}, z'_{j''} \rangle$.
- If $v = z'_{j''}$ with $j'' \neq j'$, $S_{j'} \cap S_{j''} = \emptyset$, and there is no $S_{j'''}$ such that $S_{j'} \cap S_{j'''} \neq \emptyset$ and $S_{j''} \cap S_{j'''} \neq \emptyset$, then let $x_{i'} \in S_{j'}$ and $x_{i''} \in S_{j''} \setminus \{x_i\}$, and choose $\widetilde{P} = \langle S_{j',\ell}, x_{i',\ell}, y_{i'}, y_{i''}, x_{i'',\ell}, S_{j'',\ell}, z_{j''}, z'_{j''} \rangle$. ◀

▶ **Lemma 9.** *Given a MEG-Set M' of G, we can compute in polynomial time a MEG-Set M of G such that $|M| \leq |M'|$ and, for every $\ell = 1, \ldots, k$, the set $\mathcal{S}_\ell = \{S_j \in \mathcal{S} \mid S_{j,\ell} \in M\}$ is a set cover of $\mathcal{I}$.*

Proof. Let M'' be a minimal MEG-Set of G that is obtained from M' by possibly discarding some of the vertices. Clearly, $|M''| \leq |M'|$ and M'' can be computed in polynomial time. Moreover, by Lemma 8, for every $i = 1, \ldots, \eta$ and every $\ell = 1, \ldots, k$, M'' contains $x_{i,\ell}$ or some $S_{j,\ell}$ such that S_j covers x_i. We compute M from M'' by replacing each $x_{i,\ell} \in M''$ with $S_{j,\ell}$, where $S_j \in \mathcal{S}$ is any set that covers x_i. As a consequence, for every $\ell = 1, \ldots, k$, the set $\mathcal{S}_\ell = \{S_j \in \mathcal{S} \mid S_{j,\ell} \in M\}$ is a set cover of $\mathcal{I}$. Moreover, since M'' contains all vertices in L by Lemma 1, so does M. Then, Lemma 7 implies that M is a MEG-Set of G. ◀

▶ **Lemma 10.** *Let α, c, and ε be constants of choice satisfying $\alpha > 0$, $c \geq 1$, and $0 < \varepsilon \leq \frac{1}{4c^2(\alpha+1)^2}$. Any polynomial-time $(\alpha \ln n)$-approximation algorithm for the minimum MEG-Set problem implies the existence of a polynomial-time $((2\alpha c + \sqrt{\varepsilon}) \ln \eta)$-approximation algorithm for Set Cover instances with η items and $h \leq \eta^c$ sets.*

Proof. Consider an instance $\mathcal{I} = \langle X, \mathcal{S} \rangle$ of Set Cover with $|X| = \eta$ and $|\mathcal{S}| = h \leq \eta^c$. Let h^* be the size of an optimal set cover of $\mathcal{I}$.

In the rest of the proof, we assume w.l.o.g. that $N \geq 4$, $\eta \geq 2^{1/\varepsilon}$, and $h^* \geq \frac{4\alpha}{\varepsilon}$. Indeed, if any of the above three conditions do not hold, we can solve $\mathcal{I}$ in polynomial time using a brute-force search.

We now construct the graph G with $n = (k+2)N \leq N^2$ vertices by making $k = N - 2$ copies of H. Next, we run the $(\alpha \ln n)$-approximation algorithm to compute a MEG-Set M' of G, and we use Lemma 9 to find a MEG-Set M with $|M| \leq |M'|$ that contains k set covers $\mathcal{S}_1, \ldots, \mathcal{S}_k$ in polynomial time. Among these k set covers, we output one $\mathcal{S}'$ of minimum size.

To analyze the approximation ratio of the above algorithm, let M^* be an optimal MEG-Set of G. Lemma 7 ensures that $|M^*| \leq |L| + kh^* = N + kh^*$, and hence:

$$|M| \leq |M'| \leq (\alpha \ln n)|M^*| \leq \alpha(N + kh^*) \ln n = \alpha(N + kh^*) \ln N^2 = 2\alpha k h^* \ln N + 2\alpha N \ln N.$$

Therefore, we have:

$$\begin{aligned} |\mathcal{S}'| &\leq \frac{|M|}{k} \leq 2\alpha h^* \ln N + \frac{2\alpha N \ln N}{k} \leq 2\alpha h^* \ln N + 4\alpha \ln N = \left(2\alpha + \frac{4\alpha}{h^*}\right) h^* \ln N \\ &\leq (2\alpha + \varepsilon) h^* \ln N \leq (2\alpha + \varepsilon) h^* \ln(h + \eta) \leq (2\alpha + \varepsilon) h^* \ln(2\eta^c) \\ &\leq (2\alpha + \varepsilon) h^* (1 + \varepsilon) c \ln \eta = (2\alpha c + \varepsilon c(2\alpha + \varepsilon + 1)) h^* \ln \eta \leq (2\alpha c + \sqrt{\varepsilon}) h^* \ln \eta, \end{aligned}$$

where we used $2\alpha + \varepsilon + 1 \leq 2(\alpha + 1) \leq \frac{1}{c\sqrt{\epsilon}}$, which follows from our hypothesis $\varepsilon \leq \frac{1}{4c^2(a+1)^2}$. ◀

It is known that, unless $\mathsf{P} = \mathsf{NP}$, there exists a constant $c \geq 1$ such that all polynomial-time approximation algorithms for instances of Set Cover having η items and $h \leq \eta^c$ sets must have an approximation factor of $(1 - o(1)) \ln \eta$ [18, 4]. Then, Lemma 10 with any constant $\alpha < \frac{1}{2c}$ and $\varepsilon = (\frac{1}{2} - \alpha c)^2$ implies that no polynomial-time $(\alpha \ln n)$-approximation algorithm for MEG-Set can exist, unless $\mathsf{P} = \mathsf{NP}$.

Similarly, we can choose $c = 1$ [5] to rule out any polynomial-time algorithm with an approximation of $\alpha \ln n$ for MEG-Set, where α is a constant strictly smaller than $\frac{1}{2}$, unless $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{O(\log \log n)})$.

▶ **Theorem 11.** *All polynomial-time approximation algorithms for MEG-Set must have an approximation factor of $\Omega(\log n)$, unless $\mathsf{P} = \mathsf{NP}$. Moreover, unless $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{O(\log \log n)})$, the approximation factor must be at least $(\frac{1}{2} - \varepsilon) \ln n$ for any constant $\varepsilon > 0$.*

5 Approximation algorithms

This section is devoted to designing our approximation algorithms. It is convenient to work on a generalization of the MEG-Set problem, in which each edge e of G has an associated non-negative *weight* $w(e)$ and shortest paths are computed w.r.t. these weights. Moreover, each vertex v of G has a binary cost $c(v) \in \{0, 1\}$, and the input additionally specifies a subset E' of the edges in E. The goal is to find a set of vertices M that monitors all edges in E' and has minimum cost, defined as $c(M) = \sum_{v \in M} c(v)$. We name this generalization GMEG-Set.

Notice that the case in which all vertex costs are 1, all edges weights are 1, and $E' = E$ matches the standard definition of MEG-Set, hence any approximation algorithm for GMEG-Set (whose approximation ratio is computed w.r.t. the cost of the returned set) also provides an approximation algorithm for MEG-Set.

One can observe that a GMEG-Set exists if and only if, for every edge $(u, v) \in E$, the shortest path between u and v is unique and consists of the sole edge (u, v). Indeed, if this condition is met, the set V is a trivial solution; otherwise, no subset of vertices can monitor edge (u, v). As a consequence, in the rest of this section, we restrict ourselves to instances of GMEG-Set that satisfy the above property, which can be checked in polynomial time.

In the rest of this section, we first show a simple generalization of the $O(\sqrt{n \log n})$-approximation algorithm for MEG-Set [2, 1, 7] to GMEG-Set, and then we use it as a building block to develop a more involved algorithm that provides good approximate solutions for GMEG-Set on hereditary graph classes that admit small balanced vertex-separators.

5.1 A simple approximation algorithm for GMEG-Set

We start with the description of an $\sqrt{2c(M^*) \ln n}$-approximation algorithm for GMEG-Set, where $c(M^*)$ denotes the cost of an optimal GMEG-Set, that will be useful in the sequel.

For technical convenience, we assume that all optimal solutions to the GMEG-Set instance select at least two vertices with non-zero cost.[4]

Let Z be the set of all vertices of cost 0. Observing that it is possible to check in polynomial time whether an edge of E' is monitored by Z, we construct an instance of Set Cover in which the items are exactly the edges in E' that are not monitored by Z. The

[4] Otherwise, there is at most one vertex with non-zero cost in an optimal solution, and such a vertex can be guessed in polynomial time.

instance contains a set S_p for each (unordered) pair $p = \{u, v\}$ of distinct vertices u, v in $V \setminus Z$. The set S_p contains all items e that are monitored by $Z \cup p$ (notice that an item in S_p is not necessarily monitored by p).

Next, we compute a $(\ln |E'|)$-approximate solution $\mathcal{S}'$ for such an instance in polynomial time. This can be done by solving instances up to a constant size by brute force, while larger instances can be handled using the well-known greedy algorithm for Set Cover [15], which has an approximation ratio of $\ln \eta - \ln \ln \eta + O(1)$, where η is the number of items [19]. We return the GMEG-Set M obtained by selecting all vertices in Z together with all vertices involved in at least one pair p such that $S_p \in \mathcal{S}'$, i.e., $M = Z \cup \bigcup_{S_p \in \mathcal{S}'} p$.

Observe that, if M^* is an optimal GMEG-Set, then the collection containing all sets S_p for all unordered pairs p of vertices in $M^* \setminus Z$ is a feasible solution $\mathcal{S}''$ to the Set Cover instance. Since $c(v) \in \{0, 1\}$ for each $v \in V(G)$, if $\mathcal{S}^*$ is a minimum-cost set cover, we have $|\mathcal{S}^*| \leq \binom{|M^* \setminus Z|}{2} < \frac{|M^* \setminus Z|^2}{2} = \frac{c(M^*)^2}{2}$. Then:

$$c(M) \leq 2 \cdot |\mathcal{S}'| \leq 2 \cdot |\mathcal{S}^*| \cdot \ln |E'| < c(M^*)^2 \ln |E'| \leq 2c(M^*)^2 \ln n.$$

We have proved the following lemma.

▶ **Lemma 12.** *There exists a polynomial-time $(2c(M^*) \ln n)$-approximation algorithm for GMEG-Set.*

By observing that a trivial upper bound on the approximation ratio of M is $\frac{c(V)}{c(M^*)}$, we have that the overall approximation ratio is upper bounded by $\min\left\{2c(M^*) \ln n, \frac{c(V)}{c(M^*)}\right\} \leq \sqrt{2c(V) \ln n}$, since the above minimum is maximized when $c(M^*) = \sqrt{\frac{c(V)}{2 \ln n}}$.

▶ **Lemma 13.** *There exists a polynomial-time $\sqrt{2c(V) \ln n}$-approximation algorithm for GMEG-Set.*

Lemma 13 implies the existence of a polynomial-time $O(\sqrt{n \log n})$-approximation algorithm for MEG-Set because $c(V) = n$.

5.2 An approximation algorithm based on balanced graph separators

In this section, we present our main positive result. We start by giving some preliminary definitions and proving some useful structural properties of GMEG-Sets.

5.2.1 Some structural properties

Fix some instance of GMEG-Set with graph G, vertex costs $c(\cdot)$, edge weights $w(\cdot)$, and set of edges to be monitored E'. We refer to a subset of vertices of G as a *cluster* and, given a cluster C, we define the *boundary* δC of C as $N_G(C) \setminus C$. We say that a path π between two vertices u, v in G is a *C-bypass* if all the following conditions are satisfied: (i) $u, v \in \delta C$, (ii) π is a shortest path in G, (iii) π consists of at least two edges, and (iv) all internal vertices of π belong to $V \setminus (C \cup \delta C)$. Notice that the existence of a C-bypass between two vertices u, v in G implies that $(u, v) \notin E$.

Given a cluster C, we define the *projection* of C as the following instance of GMEG-Set. The instance's graph H has a vertex set consisting of all vertices in $C \cup \delta C$, where vertices in C retain their original cost in G, while vertices in δC have a cost of 0. The edge set of H consists of all edges in $G[C \cup \delta C]$ with their original weight in G, plus some additional *projection edges*. More precisely, for each pair of distinct vertices $u, v \in \delta C$ with at least

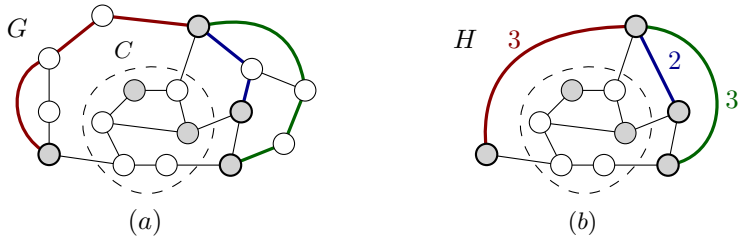


Figure 3 (a) The graph G of an instance of GMEG-Set and a cluster C. Vertices in the boundary δC are shown in bold. Three C-bypasses are highlighted in red, blue, and green, respectively. All edge weights are unitary and are not shown. (b) The graph of the projection H of C. There are three projection edges, highlighted in red, blue, and green, corresponding to the C-bypass shown with the same color in (a). The number next to a projection edge represents its weight, while non-projection edges have unit weight (not shown). Gray vertices form a GMEG-Set of H. The vertices in the boundary are shown in bold and have a cost of 0. Notice how the gray vertices also monitor all non-projection edges in G.

one C-bypass between them, we add a *projection edge* $e = (u, v)$ of weight $d_G(u, v)$, and we associate e with an arbitrarily selected C-bypass b_e between u and v. We refer to H as the *projection graph* of C, and we use E_C to denote the set of all non-projection edges of H. The edges of H to be monitored are all those in $E(H) \cap E'$. See Figure 3 for an example.

We provide different structural properties of walks in G and the projection graph H of a cluster C. To distinguish between the weights of such walks, we will add the graph as a subscript to the weight function.

▶ **Lemma 14.** *Let C be a cluster, and let H be the projection graph of C. The following claims hold for any $u, v \in C \cup \delta C$.*

(a) *For every walk π between u and v in H, there exists a walk π' between u and v in G such that $w_H(\pi) = w_G(\pi')$ and $E(\pi) \cap E_C = E(\pi') \cap E_C$;*

(b) *For every shortest path π between u and v in G, there exists a walk π' between u and v in H such that $w_G(\pi) = w_H(\pi')$ and $E(\pi) \cap E_C = E(\pi') \cap E_C$.*

Proof. We start by proving (a). Given a walk π between u and v in H, we build the walk π' from u to v in G from π by replacing each projection edge $e \in \pi$ with its associated C-bypass b_e. By construction, $E(\pi) \cap E_C = E(\pi') \cap E_C$. Moreover, for each edge $e \in E_C$, $w_H(e) = w_G(e)$ by construction, while for each projection edge e, $w_H(e) = w_G(b_e)$. Therefore, $w_H(\pi) = w_G(\pi')$.

We now prove (b). Given a shortest path π between u and v in G, we decompose π into an alternating sequence of walks $\pi_1, \widehat{\pi}_1, \pi_2, \widehat{\pi}_2, \ldots, \pi_k$, where π_i is a maximal subpath of π that contains only edges in E_C, and $\widehat{\pi}_i$ is a maximal subpath of π that contains only edges not in E_C. Let u_i, v_i be the endvertices of $\widehat{\pi}_i$. We build a walk π' from u_i to v_i in H from π by replacing each $\widehat{\pi}_i$ with the projection edge $e_i = (u_i, v_i)$ associated with a C-bypass b_{e_i} between u_i and v_i. By construction, $E(\pi) \cap E_C = E(\pi') \cap E_C$. Moreover, for each edge $e \in E_C$, $w_H(e) = w_G(e)$ by construction. Now, since π is a shortest path from u to v in G, $\widehat{\pi}_i$ is indeed a C-bypass in G, which implies $w_H(e_i) = w_G(\widehat{\pi}_i)$. Therefore, $w_G(\pi) = w_H(\pi')$. ◀

▶ **Corollary 15.** *Let C be a cluster, and let H be the projection graph of C. The following claims hold for any $u, v \in C \cup \delta C$.*

(a) *For every shortest path π between u and v in H, there exists a shortest path π' between u and v in G such that $w_H(\pi) = w_G(\pi')$ and $E(\pi) \cap E_C = E(\pi') \cap E_C$;*

(b) *For every shortest path π between u and v in G, there exists a shortest path π' between u and v in H such that $w_G(\pi) = w_H(\pi')$ and $E(\pi) \cap E_C = E(\pi') \cap E_C$.*

Proof. We start by proving (a). Let π be a shortest path between u and v in H. By Lemma 14 (a), there exists a walk π' between u and v in G such that $w_H(\pi) = w_G(\pi')$ and $E(\pi) \cap E_C = E(\pi') \cap E_C$. We now argue that π' is actually a shortest path in G. Suppose, towards a contradiction, that there exists some path π'' between u and v in G such that $w_G(\pi'') < w_G(\pi')$. By Lemma 14 (b), this implies the existence of a walk π''' between u and v in H such that $w_H(\pi''') = w_G(\pi'')$, yielding the contradiction $w_H(\pi) = w_G(\pi') > w_G(\pi'') = w_H(\pi''')$.

We now prove (b). Let π be a shortest path between u and v in G. By Lemma 14 (b), there exists a walk π' between u and v in H such that $w_G(\pi) = w_H(\pi')$ and $E(\pi) \cap E_C = E(\pi') \cap E_C$. We now argue that π' is actually a shortest path in H. Suppose, towards a contradiction, that there exists some path π'' between u and v in H such that $w_H(\pi'') < w_H(\pi')$. By Lemma 14 (a), this implies the existence of a walk π''' between u and v in G such that $w_G(\pi''') = w_H(\pi'')$, yielding the contradiction $w_G(\pi) = w_H(\pi') > w_H(\pi'') = w_G(\pi''')$. ◀

▶ **Corollary 16.** *Let C be a cluster, and let H be the projection graph of C. Let $e \in E_C$ be a non-projection edge in H, and let u, v be a pair of vertices in H. Edge e is monitored by the pair u, v in G if and only if it is monitored by u, v in H.*

Proof. Assume that e is monitored by the pair u, v in G, and let π be a shortest path between u and v in G. By Corollary 15 (b), there exists a shortest path π' between u and v in H that traverses e. We now argue that *all* shortest paths between u and v in H contain e. Indeed, if H contained a shortest path π'' between u and v avoiding e, then Corollary 15 (a) would imply the existence of a shortest path between u and v avoiding e in G, i.e., u, v would not monitor e in G.

Assume now that e is monitored by the pair u, v in H, and let π be a shortest path between u and v in H. By Corollary 15 (a), there exists a shortest path π' between u and v in G that traverses e. We now argue that *all* shortest paths between u and v in G contain e. Indeed, if G contained a shortest path π'' between u and v avoiding e, then Corollary 15 (b) would imply the existence of a shortest path between u and v avoiding e in H, i.e., u, v would not monitor e in H. ◀

▶ **Lemma 17.** *Let C be a cluster, and let M be a GMEG-Set of G. Then, $(M \cap C) \cup \delta C$ is a GMEG-Set for the projection of C.*

Proof. Consider any edge $e \in E_C$ that belongs to E', let u', v' be a pair of vertices in M that monitors e in G, and let π be a shortest path from u' to v' in G. Let u (resp. v) be the last vertex in $(M \cap C) \cup \delta C$ encountered during a traversal of the subpath π_u (resp. π_v) of π from u' (resp. v') to the closest endvertex of e. Observe that u (resp. v) exists since either $u' \in C \cup \delta C$, or π_u (resp. π_v) has the endvertex u' (resp. v') that is not in $C \cup \delta C$ and the other endvertex in $C \cup \delta C$, hence it must traverse at least one vertex in δC.

Since the subpath of π from u to v contains edge e, the pair u, v monitors e in G. Hence, by Corollary 16, u, v monitors e in the projection graph of C. ◀

▶ **Lemma 18.** *Let C be a cluster. Any GMEG-Set for the projection of C monitors all edges in $E_C \cap E'$ in G.*

Proof. Let M be a GMEG-Set for the projection of C, and let H be the corresponding graph. We consider a generic edge $e \in E_C \cap E'$ and we prove that e is also monitored by M in G. Let $u, v \in M$ be a pair of vertices that monitors e in H. By Corollary 16, the pair u, v monitors e in G. ◀

5.2.2 Algorithm description

In this section, we describe our main approximation algorithm that, as we will show, computes better than the $(\sqrt{2c(V)\ln n})$-approximate solutions for the class of hereditary graphs that admit small balanced vertex-separators. Before delving into the details of the algorithm, we provide some basic definitions.

▶ **Definition 19.** *Given $\alpha \in (0, \frac{1}{2})$, an* α-balanced vertex-separator *of a graph $G = (V, E)$ is a partition of V into three non-empty sets S, A, B, such that (i) each path from a vertex in A to a vertex in B traverses at least one vertex in S, and (ii)* $\min\{|A|, |B|\} \geq \alpha|V|$. *The size of the separator is $|S|$.*

We say that a class $\mathcal{G}$ of graphs has *efficiently computable α-balanced separators* of size $O(n^\beta)$ if (i) $\mathcal{G}$ is hereditary, i.e., given a graph $G \in \mathcal{G}$, all induced subgraphs of G are in $\mathcal{G}$, (ii) there are constants $n_0, \alpha > 0$, $\beta_0 \geq 1$, and $\beta \geq 0$ such that any graph $G \in \mathcal{G}$ with $n \geq n_0$ vertices admits an α-balanced vertex-separator of size at most $\beta_0 n^\beta$, and (iii) such a separator can be found in polynomial time. We say that a graph is α-non-separable if it admits no α-balanced vertex-separator. Similarly, we say that a cluster C of G is α-non-separable if $G[C]$ is α-non-separable. Notice that if a class of graphs $\mathcal{G}$ has efficiently computable α-balanced separators, $G \in \mathcal{G}$, and C is an α-non-separable cluster of G, then $|C| = O(1)$.

We consider a class $\mathcal{G}$ with efficiently computable α-balanced separators of size $\beta_0|V(G)|^\beta$ for some constants $\beta_0 \geq 1$ and $\beta \geq 0$, and we restrict our attention (w.l.o.g.) to instances of GMEG-Set whose input graph $G = (V, E)$ belongs to $\mathcal{G}$ and satisfies $c(V) \geq 2$.[5] Our algorithm works in two phases.

First phase. In the first phase, we compute a hierarchical decomposition of G into clusters that are pairwise vertex-disjoint. We note that such a decomposition will not result in a partition of the vertices of G, i.e., some vertices will not be assigned to any of the clusters, and we refer to the vertices that are left unassigned as *crossing vertices*. Our decomposition relies on recursively computing α-balanced vertex-separators of G and guarantees that any path between vertices of different clusters traverses at least one crossing vertex. It follows that all the vertices in the boundaries of the returned clusters are crossing vertices.

Let G' be the induced subgraph of G that contains all (and only) the vertices having non-zero cost. We build a rooted tree T in which each vertex u of T is associated with some subgraph G_u of G' and with an instance of GMEG-Set H_u. The root r of T is associated with $G_r = G'$ and with the projection H_r of $V(G_r)$ (w.r.t. the input instance of GMEG-Set). A generic vertex u of T is a leaf if H_u admits a solution of cost 0, or if G_u is α-non-separable. Otherwise, u has exactly two children a, b in T corresponding to the subgraphs $G_a = G_u[A]$ and $G_b = G_u[B]$ induced by the sets A, B of an α-balanced vertex-separator S_u, A, B of G_u. The instances of GMEG-Set H_a and H_b are the projections of $V(G_a)$ and $V(G_b)$ w.r.t. the instances H_u, respectively. See Figure 4 for an example.

Second phase. In the second phase of our algorithm, we visit the tree in a bottom-up fashion and, for each examined vertex u, we compute a $\rho(h_u, n_u)$-approximate solution for the projection H_u of $V(G_u)$. Here, $\rho(h_u, n_u)$ is a function that depends on the number $n_u = |V(G_u)|$ of vertices in G_u and, possibly, on the height h_u of u in T.[6] Observe that all vertices in G_u have cost 1 w.r.t. the cost function c_u of H_u, i.e., $n_u = c_u(V(G_u))$.

[5] Otherwise, when $c(V) \leq 1$, an optimal solution consists either of all vertices of cost 0, or of all vertices of cost 0 plus the sole vertex of cost 1.

[6] The height of vertex u in T is the height of the subtree of T rooted in u, i.e., the maximum number of edges in the longest path from u to a (not necessarily proper) descendant of u.

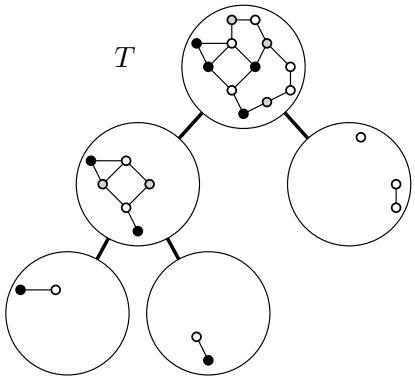


Figure 4 The tree T representing a possible hierarchical decomposition of the graph shown in the root vertex for the parameters $\beta_0 = 1$, $\beta = \frac{1}{2}$, $\alpha = \frac{1}{4}$. Vertices in a separator are shown in gray in the corresponding cluster. Black vertices in a leaf cluster u of T form an optimal solution to instance H_u (not shown) when combined with the vertices of cost 0 resulting from the projection. Observe that the instance corresponding to the right child of the root admits a solution of cost 0. Gray and black vertices in an internal cluster u form a feasible GMEG-Set of the corresponding instance H_u (not shown) when combined with the vertices of cost 0.

In detail, if u is a leaf in T, then either there exists an optimal GMEG-Set of H_u with cost 0, or $|V(G_u)| = O(1)$, and we can find an optimal GMEG-Set of H_u in constant time by brute force (possibly both). Regardless of the case, we have an optimal solution for H_u, i.e., a $\rho(0, n_u)$-approximation for $\rho(0, n_u) = 1$.

When the examined vertex u of T is a generic internal vertex at height h having a and b as children, we compute a GMEG-Set M_u for H_u by choosing the set of smallest cost between (i) a GMEG-Set M'_u obtained as the union of a $\rho(h_u - 1, n_a)$-approximate GMEG-Set M_a for H_a with a $\rho(h_u - 1, n_b)$-approximate GMEG-Set M_b for H_b, and (ii) a $(2c_u(M^*) \ln |V(G_u)|)$-approximate solution computed using the algorithm of Section 5.1 (see Lemma 12), where M^* denotes an optimal solution for H_u and c_u is the cost function of H_u. The next lemma provides an upper bound on the cost of $c_u(M'_u)$.

▶ **Lemma 20.** *Let u be an internal vertex at height h_u with children a and b. If $\rho(h_u - 1, x)$ is monotonically non-decreasing w.r.t. x, M_a is a $\rho(h_u - 1, n_a)$-approximate GMEG-Set for H_a, and M_b is a $\rho(h_u - 1, n_b)$-approximate GMEG-Set for H_b, then M'_u is a GMEG-Set for H_u with $c_u(M'_u) \leq |S_u| + c_u(M^*_u)\rho(h_u - 1, (1 - \alpha)n_u)$, where M^*_u is an optimal solution for H_u.*

Proof. Let $M^*_a = (M^* \cap V(G_a)) \cup \delta V(G_a)$ (resp. $M^*_b = (M^* \cap V(G_b)) \cup \delta V(G_b)$), where the boundary is w.r.t. the instance H_u. By Lemma 17, M^*_a and M^*_b are GMEG-Set for H_a and H_b, respectively. Then, $c_a(M^*_a) = c_a(M^* \cap V(G_a)) + c_a(\delta V(G_a)) = c_a(M^* \cap V(G_a)) = c_u(M^* \cap V(G_a))$, and a similar derivation shows that $c_b(M^*_b) = c_u(M^* \cap V(G_b))$.

Since M_a and M_b are a $\rho(h_u - 1, n_a)$- and a $\rho(h_u - 1, n_b)$-approximation of H_a and H_b, respectively, we have $c_a(M_a) \leq c_u(M^* \cap V(G_a))\rho(h_u - 1, n_a)$ and $c_b(M_b) \leq c_u(M^* \cap V(G_b))\rho(h_u - 1, n_b)$.

Let E'_u be the set of edges to be monitored in H_u. By Lemma 18, M_a monitors all edges of E'_u in $G_u[V(G_a) \cup \delta V(G_a)]$, and M_b monitors all edges of E'_u in $G_u[V(G_b) \cup \delta V(G_b)]$; therefore, $M_a \cup M_b$ is a GMEG-Set for H_u. Since $M_a \cup M_b \subseteq M'_u$, we have that M'_u is a GMEG-Set of H_u of cost:

$$\begin{aligned} c_u(M') &\leq c_u(M_a \cup M_b) \leq c_u(S_u) + c_u(M_a \setminus S) + c_u(M_b \setminus S) \leq |S_u| + c_a(M_a) + c_b(M_b) \\ &\leq |S_u| + c_u(M_a^* \cap V(G_a))\rho(h_u - 1, n_a) + c_u(M_b^* \cap V(G_b))\rho(h_u - 1, n_b) \\ &\leq |S_u| + \big(c_u(M_a^* \cap V(G_a)) + c_u(M_b^* \cap V(G_b))\big)\rho(h_u - 1, (1-\alpha)n_u) \\ &\leq |S_u| + c_u(M^*)\rho(h_u - 1, (1-\alpha)n_u). \end{aligned}$$ ◀

The following two lemmas upper bound the approximation ratio achieved by M_u for the instance H_u.

▶ **Lemma 21.** *Let $\rho(h, x) = x^{\beta/2} \cdot \frac{\sqrt{2\beta_0 \ln n}}{1-(1-\alpha)^{\beta/2}}$. If $\beta > 0$, then M_u is a $\rho(h_u, n_u)$-approximate GMEG-Set for H_u.*

Proof. The proof is by induction on h_u. The claim is trivially true for $h_u = 0$ since $\rho(0, n_u) = 1 \leq \sqrt{2 \ln 2} \leq \frac{\sqrt{n_u \beta_0 \ln n}}{1-(1-\alpha)^{\beta/2}}$. Therefore, we assume that the claim holds for $h_u - 1 \geq 0$ and we prove that it holds for h_u. Consider a vertex u at height h_u, and let a and b be its two children, and let S_u be the corresponding separator of G_u. Let M^* be an optimal GMEG-Set for H_u. By Lemma 20, we have:

$$c(M) \leq |S_u| + c(M^*)\rho(h_u - 1, (1-\alpha)n_u) \leq \beta_0 n_u^\beta + c(M^*)(1-\alpha)^{\beta/2} n_u^{\beta/2} \cdot \frac{\sqrt{2\beta_0 \ln n}}{1-(1-\alpha)^{\beta/2}}.$$

Since the returned solution is the one of minimum cost between M'_u and a solution with approximation ratio $2c(M^*) \ln |V(G_u)| \leq 2c(M^*) \ln n$, the resulting approximation ratio is:

$$\begin{aligned} &\min\left\{\frac{\beta_0 n_u^\beta}{c(M^*)} + (1-\alpha)^{\beta/2} n_u^{\beta/2} \frac{\sqrt{2\beta_0 \ln n}}{1-(1-\alpha)^{\beta/2}}, 2c(M^*) \ln n\right\} \\ &\leq \min\left\{\frac{\beta_0 n_u^\beta}{c(M^*)}, 2c(M^*) \ln n\right\} + (1-\alpha)^{\beta/2} n_u^{\beta/2} \frac{\sqrt{2\beta_0 \ln n}}{1-(1-\alpha)^{\beta/2}} \\ &\leq \sqrt{2\beta_0 \ln n} \cdot n_u^{\beta/2} + (1-\alpha)^{\beta/2} n_u^{\beta/2} \frac{\sqrt{2\beta_0 \ln n}}{1-(1-\alpha)^{\beta/2}} \\ &= \sqrt{2\beta_0 \ln n} \cdot n_u^{\beta/2} \left(1 + \frac{(1-\alpha)^{\beta/2}}{1-(1-\alpha)^{\beta/2}}\right) = n_u^{\beta/2} \cdot \frac{\sqrt{2\beta_0 \ln n}}{1-(1-\alpha)^{\beta/2}} \end{aligned}$$

where, in the second inequality, we use the fact that $\min\left\{\frac{\beta_0 n_u^\beta}{c(M^*)}, 2c(M^*) \ln n\right\}$ is maximized for $\frac{\beta_0 n_u^\beta}{c(M^*)} = 2c(M^*) \ln n$, i.e., when $c(M^*) = \sqrt{\frac{\beta_0 n_u^\beta}{2 \ln n}}$. ◀

▶ **Lemma 22.** *Let $\rho(h, x) = (h+1)\sqrt{2\beta_0 \ln n}$. If $\beta = 0$, then M_u is a $\rho(h_u, n_u)$-approximate GMEG-Set for H_u.*

Proof. The proof is by induction on h. The claim is trivially true for $h = 0$ since $\rho(0, n_u) = 1 \leq \sqrt{2 \ln 2} \leq (h_u + 1)\sqrt{n_u \beta_0 \ln n}$. Therefore, we assume that the claim holds for $h_u - 1 \geq 0$ and we prove that it holds for h_u. Consider a vertex u at height h_u, let a and b be its two children, and let S_u be the corresponding separator of G_u. Let M^* be an optimal solution for H_u. By Lemma 20, we have:

$$c(M) \leq |S_u| + c(M^*)\rho(h_u - 1, (1-\alpha)n) \leq \beta_0 n_u^\beta + c(M^*) h_u \sqrt{2\beta_0 \ln n}.$$

Since the returned solution is the one of minimum cost between M'_u and a solution with approximation ratio $2c(M^*)\ln|V(G_u)| \leq 2c(M^*)\ln n$, the resulting approximation ratio is:

$$\begin{aligned}\min\left\{\frac{\beta_0}{c(M^*)} + h_u\sqrt{2\beta_0\ln n}, 2c(M^*)\ln n\right\} &\leq \min\left\{\frac{\beta_0}{c(M^*)}, 2c(M^*)\ln n\right\} + h_u\sqrt{2\beta_0\ln n}\\ &\leq \sqrt{2\beta_0\ln n} + h_u\sqrt{2\beta_0\ln n} = (h_u+1)\sqrt{2\beta_0\ln n},\end{aligned}$$

where we used the fact that $\min\left\{\frac{\beta_0}{c(M^*)}, 2c(M^*)\ln n\right\}$ is maximized for $c(M^*) = \sqrt{\frac{\beta_0}{2\ln n}}$. ◀

We can finally state the main results of this section.

▶ **Theorem 23.** *There exists a polynomial-time approximation algorithm for GMEG-Set with an approximation ratio of $O(c(V)^{\beta/2}\sqrt{\log n})$ if $\beta > 0$ and an approximation ratio of $O(\log c(V)\cdot\sqrt{\log n})$ if $\beta = 0$.*

Proof. We start by arguing that M_r achieves the claimed approximation ratios for H_r. When $\beta > 0$, this follows directly from Lemma 21 and from the fact that G_r has $n_r = c(V)$ vertices. Regarding the case $\beta = 0$, if u is the parent of v, then $n_v \leq (1-\alpha)n_u$. Then, the tree T has height at most $\left\lceil \log_{\frac{1}{1-\alpha}} n_r \right\rceil = \left\lceil \frac{\log c(V)}{\log \frac{1}{1-\alpha}} \right\rceil$, and Lemma 22 implies that the achieved approximation is at most $\left(1 + \left\lceil \frac{\log c(V)}{\log \frac{1}{1-\alpha}} \right\rceil\right)\cdot\sqrt{2\beta_0\ln n}$.

To obtain the claimed approximations for the input instance of GMEG-Set, let Z be the set of vertices having cost 0 in such instance, and consider $M = M_r \cup Z$. Since $c(M) = c(M_r)$, $c(V) = c(V(G'))$, and $n \geq n_r$, the claim on the approximation trivially holds, and we only need to argue that M is indeed a GMEG-Set.

By Lemma 18, M_r monitors, in G, all edges of E' that belong to $V(G') \cup \delta V(G')$. Hence, if $V(G') \cup \delta V(G') = V(G)$, we are done. Otherwise, let $A = V(G) \setminus (V(G') \cup \delta V(G'))$, and let H_A be the projection of A. Since $A \subseteq Z$, the set Z is a GMEG-Set for H_A, and by Lemma 18, Z monitors, in G, all edges of E' that belong to $V(A) \cup \delta V(A)$. Hence, $A \cup M_r \subseteq Z \cup M_r$ monitors all edges in E', i.e., it is a GMEG-Set. ◀

The above theorem implies the following approximability results for GMEG-Set (and for MEG-Set, where $c(V) = n$):

▶ **Corollary 24.** *GMEG-Set is approximable in polynomial time on:*

- *planar graphs, with an approximation ratio of $O(c(V)^{\frac{1}{4}}\sqrt{\log n})$ using the balanced separators of size $O(\sqrt{n})$ shown in [16];*
- *graphs with bounded treewidth, with an approximation ratio of $O(\log c(V)\sqrt{\log n})$, using the separator induced by a centroid bag of the corresponding tree decomposition [3, Sections 12.3 and 12.4];*
- *graphs with constant genus, with an approximation ratio of $O(c(V)^{\frac{1}{4}}\sqrt{\log n})$ using the balanced separators of size $O(\sqrt{n})$ shown in [11];*
- *graphs with apex $k = O(n^{1/4})$, with an approximation ratio of $O(c(V)^{\frac{1}{4}}\sqrt{\log n})$, since it is possible to find, in polynomial time, a set S of $O(k)$ vertices such that $G[V(G)\setminus S]$ is planar [14].*

References

1. Davide Bilò, Giordano Colli, Luca Forlizzi, and Stefano Leucci. On the inapproximability of finding minimum monitoring edge-geodetic sets (short paper). In Ugo de'Liguoro, Matteo Palazzo, and Luca Roversi, editors, *Proceedings of the 25th Italian Conference on Theoretical Computer Science, Torino, Italy, September 11-13, 2024*, volume 3811 of *CEUR Workshop Proceedings*, pages 219–224. CEUR-WS.org, 2024. URL: https://ceur-ws.org/Vol-3811/paper110.pdf.
2. Giordano Colli. Monitoring graph edges via shortest paths: computational complexity and approximation algorithms. Bachelor's thesis, December 2023. Publicly accessible version: https://arxiv.org/abs/2506.12021. URL: https://hdl.handle.net/20.500.12319/14489.
3. Reinhard Diestel. *Graph Theory, 5th Edition*. Graduate texts in mathematics. Springer, 2018.
4. Irit Dinur and David Steurer. Analytical approach to parallel repetition. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 624–633. ACM, 2014. doi:10.1145/2591796.2591884.
5. Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998. doi:10.1145/285055.285059.
6. Florent Foucaud, Clara Marcille, Zin Mar Myint, R. B. Sandeep, Sagnik Sen, and S. Taruni. Bounds and extremal graphs for monitoring edge-geodetic sets in graphs. *Discret. Appl. Math.*, 366:106–119, 2025. doi:10.1016/J.DAM.2024.12.032.
7. Florent Foucaud, Clara Marcille, R. B. Sandeep, Sagnik Sen, and S Taruni. Algorithms and complexity for monitoring edge-geodetic sets in graphs, 2024. doi:10.48550/arXiv.2409.19067.
8. Florent Foucaud, Pierre-Marie Marcille, Zin Mar Myint, R. B. Sandeep, Sagnik Sen, and S. Taruni. Monitoring edge-geodetic sets in graphs: Extremal graphs, bounds, complexity. In Subrahmanyam Kalyanasundaram and Anil Maheshwari, editors, *Algorithms and Discrete Applied Mathematics - 10th International Conference, CALDAM 2024, Bhilai, India, February 15-17, 2024, Proceedings*, volume 14508 of *Lecture Notes in Computer Science*, pages 29–43. Springer, 2024. doi:10.1007/978-3-031-52213-0_3.
9. Florent Foucaud, Krishna Narayanan, and Lekshmi Ramasubramony Sulochana. Monitoring edge-geodetic sets in graphs. In Amitabha Bagchi and Rahul Muthu, editors, *Algorithms and Discrete Applied Mathematics - 9th International Conference, CALDAM 2023, Gandhinagar, India, February 9-11, 2023, Proceedings*, volume 13947 of *Lecture Notes in Computer Science*, pages 245–256. Springer, 2023. doi:10.1007/978-3-031-25211-2_19.
10. Florent Foucaud, Arti Pandey, and Kaustav Paul. Characterizing optimal monitoring edge-geodetic sets for some structured graph classes, 2025. doi:10.48550/arXiv.2503.06086.
11. John R. Gilbert, Joan P. Hutchinson, and Robert Endre Tarjan. A separator theorem for graphs of bounded genus. *J. Algorithms*, 5(3):391–407, 1984. doi:10.1016/0196-6774(84)90019-1.
12. John Haslegrave. Monitoring edge-geodetic sets: Hardness and graph products. *Discret. Appl. Math.*, 340:79–84, 2023. doi:10.1016/j.dam.2023.06.033.
13. Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/JCSS.2000.1727.
14. Bart M. P. Jansen and Michal Wlodarczyk. Lossy planarization: a constant-factor approximate kernelization for planar vertex deletion. 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 900–913. ACM, 2022. doi:10.1145/3519935.3520021.
15. David S. Johnson. Approximation algorithms for combinatorial problems. In Alfred V. Aho, Allan Borodin, Robert L. Constable, Robert W. Floyd, Michael A. Harrison, Richard M. Karp, and H. Raymond Strong, editors, *Proceedings of the 5th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1973, Austin, Texas, USA*, pages 38–49. ACM, 1973. doi:10.1145/800125.804034.
16. Richard J. Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979. doi:10.1137/0136016.

17 Rongrong Ma, Zhen Ji, Yifan Yao, and Yalong Lei. Monitoring-edge-geodetic numbers of radix triangular mesh and sierpiński graphs. *Int. J. Parallel Emergent Distributed Syst.*, 39(3):353–361, 2024. doi:10.1080/17445760.2023.2294369.

18 Dana Moshkovitz. The projection games conjecture and the np-hardness of ln n-approximating set-cover. *Theory Comput.*, 11:221–235, 2015. doi:10.4086/TOC.2015.V011A007.

19 Peter Slavík. A tight analysis of the greedy algorithm for set cover. *J. Algorithms*, 25(2):237–254, 1997. doi:10.1006/JAGM.1997.0887.

20 Ao Tan, Wen Li, Xiumin Wang, and Xin Li. Monitoring edge-geodetic numbers of convex polytopes and four networks. *Int. J. Parallel Emergent Distributed Syst.*, 38(4):301–312, 2023. doi:10.1080/17445760.2023.2220143.

21 Xin Xu, Chenxu Yang, Gemaji Bao, Ayun Zhang, and Xuan Shao. Monitoring-edge-geodetic sets in product networks. *Int. J. Parallel Emergent Distributed Syst.*, 39(2):264–277, 2024. doi:10.1080/17445760.2024.2301929.

22 Igor E. Zvervich, Vadim E. Zverovich, Igor Zverovich, and Vadim Zverovich. An induced subgraph characterization of domination perfect graphs. *Journal of Graph Theory*, 20:375–395, 1995. doi:10.1002/jgt.3190200313.

Parameterized Complexity of Directed Traveling Salesman Problem

Václav Blažej
Institute of Informatics, University of Warsaw, Poland

Andreas Emil Feldmann
University of Sheffield, UK

Foivos Fioravantes
Faculty of Information Technology, Czech Technical University in Prague, Czech Republic

Paweł Rzążewski
Warsaw University of Technology, Poland
Institute of Informatics, University of Warsaw, Poland

Ondřej Suchý
Faculty of Information Technology, Czech Technical University in Prague, Czech Republic

Abstract

The Directed Traveling Salesman Problem (DTSP) is a variant of the classical Traveling Salesman Problem in which the edges in the graph are directed and a vertex and edge can be visited multiple times. The goal is to find a directed closed walk of minimum length (or total weight) that visits every vertex of the given graph at least once. In a yet more general version, Directed Waypoint Routing Problem (DWRP), some vertices are marked as terminals and we are only required to visit all terminals. Furthermore, each edge has its capacity bounding the number of times this edge can be used by a solution.

While both problems (and many other variants of TSP) were extensively investigated, mostly from the approximation point of view, there are surprisingly few results concerning the parameterized complexity. Our starting point is the result of Marx et al. [APPROX/RANDOM 2016] who proved that DTSP is W[1]-hard parameterized by distance to pathwidth 3. In this paper we aim to initiate the systematic complexity study of variants of Directed Traveling Salesman Problem with respect to various, mostly structural, parameters.

We show that DWRP is FPT parameterized by the solution size, the *feedback edge number* and the *vertex integrity* of the underlying undirected graph. Furthermore, the problem is XP parameterized by treewidth.

On the complexity side, we show that the problem is W[1]-hard parameterized by the distance to constant treedepth.

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

Keywords and phrases Directed TSP, parameterized complexity, vertex integrity, treedepth

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.15

Related Version *Full Version*: `https://arxiv.org/abs/2506.22127` [14]

Funding *Václav Blažej and Paweł Rzążewski*: The work on this article is a part of project BOBR that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 948057).
Ondřej Suchý: Co-funded by the European Union under the project Robotics and advanced industrial production (reg. no. CZ.02.01.01/00/22_008/0004590).

Acknowledgements The project was initiated at the workshop Homonolo 2023. We are grateful to the organizers and other participants for a nice and productive atmosphere.

© Václav Blažej, Andreas Emil Feldmann, Foivos Fioravantes, Paweł Rzążewski, and Ondřej Suchý;
licensed under Creative Commons License CC-BY 4.0

1 Introduction

The Traveling Salesman Problem (TSP) is possibly the most famous combinatorial optimization problem. It is one of the few problems having entire monographs devoted solely to its aspects [5, 19, 35, 53]. In this paper we focus on the directed variant, which can be formulated as follows. The input instance is a directed graph G with edges assigned positive integers (*weights*) and an integer b, and we are to decide whether there is a closed walk in G visiting every vertex *at least* once and having total weight at most b. We refer to this problem as Directed Traveling Salesman Problem (DTSP). We also consider two natural generalizations of the problem. In the first of them, Directed Subset TSP (DsTSP), we are given alongside the graph also a subset of its vertices W called *waypoints* or *terminals* and the walk is only required to visit these vertices. Obviously, an instance of DTSP can be interpreted as the instance of DsTSP by letting $W = V(G)$. In some applications it is also preferred not to traverse any (or some) of the edges of the graph too often. To this end, in Directed Waypoint Routing Problem (DWRP) (introduced in [2]), we augment the input with a positive integer for each edge specifying its *capacity*. The walk is then required to traverse each edge at most as many times as its capacity. A simple argument (provided later) shows that each edge is used at most $n := |V(G)|$ times in any solution. Hence, setting the capacities to n is equivalent to dropping the capacity constraints completely.

Our focus is on exact parameterized algorithms for the problem. Let us first summarize the known results for the undirected variant of the problem. Since Hamiltonian Cycle (HamC) is NP-hard even on planar graphs of maximum degree three [32], TSP is para-NP-hard with respect to (w.r.t.) the maximum degree of the input graph as well as w.r.t. the maximum number of visits to any vertex. As the solution size is equal to the number of vertices, the dynamic programming of Bellman [9] and Held and Karp [36] which runs in $O(2^n \cdot n^2)$ time can be considered as an FPT algorithm w.r.t. solution size. It can be also used to solve Subset TSP in $O(2^{|W|} \cdot |W|^2)$ after an initial computation of distances between all pairs of waypoints. This can be improved to $O(2^{\sqrt{|W|}\log|W|}n^{O(1)})$ if the graph is planar and the weights are bounded by a polynomial [40].

On the one hand, a single-visit variant of TSP is known to be FPT w.r.t. treewidth (see also [23]) and this result also extends to our multi-visit variant and to Waypoint Routing (WPR) [51]. On the other hand, TSP is hard on cliques with edge weights 1 and 2, shown by a simple reduction from HamC. Therefore, it is para-NP-hard w.r.t. any parameter which is constant on cliques. Moreover, already HamC is W[1]-hard w.r.t. cliquewidth [30] (while FPT w.r.t. the neighborhood diversity [43]).

Recently, the (undirected) TSP was studied from the perspective of kernelization [12, 13]. It was shown to admit a polynomial kernel w.r.t. the feedback edge number and w.r.t. distance to constant size components. Here, for a graph class $\mathcal{C}$, *distance to* $\mathcal{C}$ refers to the minimum number of vertices that must be deleted from the input graph to obtain a graph that belongs to $\mathcal{C}$. In the above case, the class contains all graphs with every connected component of size at most c, for any fixed c. TSP was shown not to admit polynomial kernel w.r.t. vertex integrity, unless NP $\subseteq$ coNP/poly, whereas Subset TSP does not admit polynomial kernel even w.r.t. distance to disjoint cycles, under the same assumption [13]. Here *vertex integrity* of a graph is the minimum k such that we can delete at most k vertices from the graph in such a way that each connected component of the resulting graph has at most k vertices.

Much less is known about the parameterized complexity of the directed case of the problem. The mentioned hardness results translate to the directed case by symmetrically orienting the input graph. While the algorithmic results do not transfer automatically to the

directed setting, it is often possible to adapt them. For example, the dynamic programming w.r.t. number of vertices [9, 36] applies also to this case. This can be again improved to $O(2^{\sqrt{|W|}\log|W|}n^{O(1)})$ for DsTSP in planar graphs [47]; also lifting the weight-restriction from the undirected case. In contrast, DWRP is NP-hard already for two waypoints [2].

Similarly, the single visit variant is FPT w.r.t. the treewidth. However, Marx et al. [48] showed, that DTSP is W[1]-hard w.r.t. the pathwidth pw of the input graph and there is no algorithm for DTSP running in $f(\text{pw}) \cdot n^{o(\text{pw})}$ time for any computable function f, unless the Exponential Time Hypothesis (ETH) [37, 38] fails. More precisely, the proof shows that this is the case even w.r.t. distance to graphs of pathwidth 3. Furthermore, a more careful analysis of the proof (see the full version for details) shows that the W[1]-hardness holds true even w.r.t. distance to pathwidth 2. Thus, the fpt-algorithm for WRP w.r.t. treewidth cannot be transferred to the directed case.

Apparently, these are the only results concerning the parameterized complexity of DTSP, and, in particular, no fpt-algorithms w.r.t. structural parameters are known for the problem.

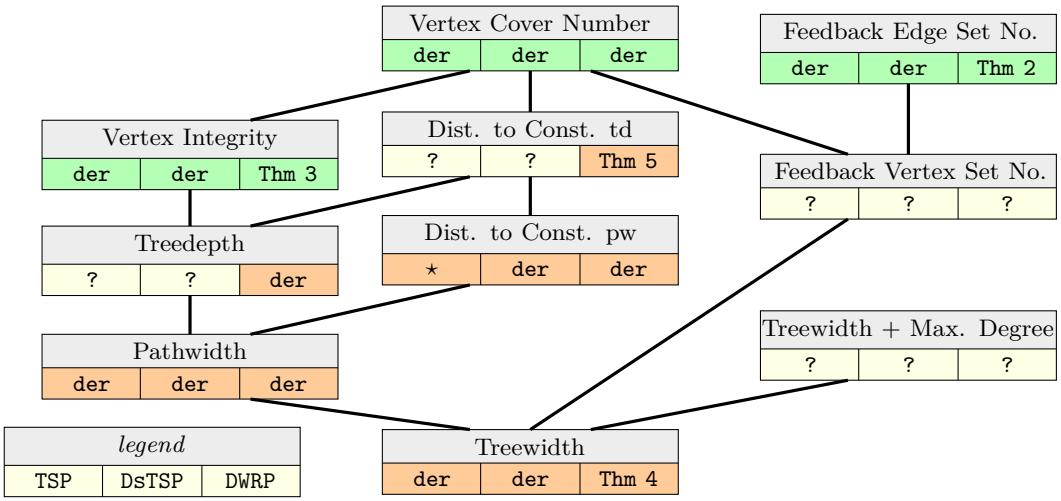


Figure 1 Complexity picture for the studied problems. Box fill shows tractability: green stands for FPT, orange for W[1]-hard and in XP, and yellow for in XP, but open whether FPT or W[1]-hard. The boxes either contain references to the corresponding theorem, or `der` if the result can be derived from the other results. All the hardness results apply even for unit weights, whereas the algorithms do not assume any restriction on the weights.

Our Contribution. We initiate the systematic study of the parameterized complexity of DTSP and its generalizations focusing on structural parameters. In particular, we concentrate on parameters measuring the structure of the underlying undirected graph of the input. We aim to fill the apparent gap in knowledge and address the lack of fpt-algorithms.

First, we revisit the standard parameterization, i.e., the parameterization by the solution size. The budget bounds the number of edges in the solution, hence also the number of vertices visited, and in particular the size of W (for any yes-instance of DTSP and DsTSP) Thus for these problems, the known single exponential algorithms [9, 36] and subexponential algorithm for planar graphs [47] show that the problem is FPT w.r.t. the budget b. For DWRP the situation is not as straightforward. Nevertheless, we show the following result.[1]

[1] Proofs of statements marked with ($\star$) can be found in the full version.

▶ **Theorem 1** (⋆). *In $e^k 4^k k^{O(\log k)} \cdot m \log m$ time we can decide whether the given m-edge instance of DWRP has a solution consisting of at most k edge occurrences.*

Next we turn to the structural parameters (see Figure 1 for an overview). The result of Marx et al. [48] shows that DTSP is already W[1]-hard w.r.t. distance to graphs of pathwidth 3, i.e., for graphs with rather restricted structure. In fact, as we show in the full version, this holds also for unweighted graphs. Therefore, in search for fpt-algorithms for the problems, the structure of the graph after removal of parameter many vertices or edges should be more restricted than having pathwidth 3. Our first positive result is w.r.t. the feedback edge number, i.e., the minimum number of edges that needs to be removed to obtain a forest.

▶ **Theorem 2.** *DWRP can be solved in $k^{O(k)} + n^{O(1)}$ time, where k is the feedback edge number of the input graph.*

Then, we present our main algorithmic result, showing that DWRP is FPT w.r.t. vertex integrity. Recall that vertex integrity allows, roughly speaking, to delete parameter many vertices so as to obtain a graph with connected components of parameter size.

▶ **Theorem 3.** *DWRP can be solved in $k^{O(k^6)} \cdot n \cdot \log^2 n(\log n + \log U)$ time, where k is the vertex integrity of the input graph and U is the maximum weight of an edge in G.*

The algorithm first guesses a part of the solution ensuring its connectivity and then employs a well structured ILP (*N-fold IP* [41]) to find the cheapest way to complete the initial guess into a solution.

We also show that, while the problems are probably not FPT w.r.t. treewidth alone, they are FPT w.r.t. treewidth combined with the bound on the number of visits to every city.

▶ **Theorem 4** (⋆). *DWRP on n-vertex graphs of treewidth t can be solved in $n^{O(t)}$ time.*

This means that all considered problems are in XP w.r.t. treewidth and, hence, w.r.t. all structural parameters considered in this paper.

We complement the positive results by showing that DWRP is W[1]-hard w.r.t. distance to constant treedepth, i.e., if the input graph becomes of constant treedepth after deleting parameter many vertices.

▶ **Theorem 5.** *DWRP is W[1]-hard with respect to modulator to constant treedepth, even if all weights are 1.*

Note that this result is incomparable to that of Marx et al. [48] in that it holds for a more restricted graph class, while it only applies to a more general problem.

Further Related Work. Another popular formulation of (Directed) TSP is that the input graph is complete and, thus, we are given a matrix of weights for all ordered pairs of vertices. Since this matrix is asymmetric for the case of directed graphs, the directed variant of TSP is often called Asymmetric TSP (ATSP). It is easy to switch from graph G to a complete graph on the same set of vertices by performing the so called *metric closure*, i.e., assigning the edge (u, v) of the complete graph the weight given by distance from vertex u to vertex v in the graph G. The weights obtained this way satisfy *triangle inequality*. Whenever this is the case, there is no reason to revisit any vertex, as it cannot make the tour shorter. However, if the matrix is not required to satisfy the triangle inequality, it makes a difference whether visiting a vertex more than once is allowed or not (single visit version).

For the single visit version with general matrices a folklore result shows that it cannot be approximated within any polynomial factor, unless P=NP [20]. For undirected case with triangle inequality it was know that there is $\frac{3}{2}$-approximation since 1976 [18], which was

later improved [33, 49, 50] up to the currently best known factor 1.4 [52] for the graphic case, where the weights originate from the distances in an unweighted graph. However, for the directed case, it was long open, whether a constant factor approximation exists for matrices satisfying the triangle inequality [4, 6]. Constant-factor approximation algorithms appeared only recently, first for unweighted directed graphs [54], later for general distances [55], with the currently best approximation factor being $22 + \varepsilon$ for any $\varepsilon > 0$ [56]. The best known lower bound on the approximability is $\frac{75}{74}$ [39].

It is surprising, given the lack of parameterized analysis for the problem, that there are several studies concerning parameterized approximation algorithm for DTSP. First, Böckenhauer [15] considered (undirected) TSP with deadlines on the latest time to visit some of the vertices and showed a 2.5-approximation algorithm in $k! \cdot n^{O(1)}$ time, where k is the number of deadlines, and complemented that by several innaproximability results. The main result of the already mentioned paper of Marx et al. [48] is a polynomial-time constant-factor approximation for DTSP in nearly embeddable graphs, where both the factor and the time depend on the actual parameters of the class of graphs considered. Bonnet et al. [17] provided a $(\log r)$-approximation for ATSP in $2^{O(\frac{n}{r})}$ time for any r. Behrendt et al. [8] considered the case that there are few edges with (significantly) asymmetric costs and presented a 2.5-approximation parameterized by the vertex cover number of the graph formed by such edges and a 3-approximation parameterized by the minimum number of asymmetric edges in a minimum arborescence of the graph.

Further results are known for a variant of DWRP where the waypoints have a prescribed order [1] and for the undirected case [3].

Another studied variant of DTSP is the Many-visit TSP, where each vertex has a prescribed number of visits and the task is to find a closed walk which visits each vertex *exactly* the given number of times. After a significant effort, an algorithm was found for the problem that is exponential only in the number of vertices, i.e., independent of the demands which can be huge [11, 42]. The problem was recently studied from parameterized perspective [45] under the name Connected Flow. Here only a subset of vertices has a prescribed number of visits, the other vertices can be visited arbitrary number of times (or not visited at all) and edges have capacities (as in DWRP). They showed that the problem is para-NP-hard w.r.t. number of vertices with demand even in unweighted graph, while FPT w.r.t. this parameter, if there are no capacities. Further, they gave a polynomial kernel w.r.t. vertex cover number, an $n^{O(\mathrm{tw})}$-time algorithm and a matching $n^{o(\mathrm{tw})}$-time ETH lower bound.

Several parameterized and fine-grained complexity studies considered the local search for the undirected single-visit TSP parameterized the size of the neighborhood to be searched [10, 16, 22, 34, 44, 46]. None of these studies seems to extend to directed graphs.

2 Notations, Preliminaries, and Basic Observations

We follow standard graph-theoretic notation [24]. Also, we refer the reader to [21] for a textbook on parameterized complexity.

(Generalized) N-fold integer programming (N-fold IP). is the problem of minimizing a separable convex objective (for us it suffices to minimize a linear objective) over a set of constraints of a specific form. Let $r \in \mathbb{N}$ and $s_i, t_i \in \mathbb{N}$ for every $i \in [N]$. The IP has $d = \sum_{i \in [N]} t_i$ variables partitioned into N so-called *bricks*. The i-th brick is denoted $x^{(i)}$

and contains t_i variables. The constraints have the following form:

$$\begin{aligned} D_1x^{(1)} + D_2x^{(2)} + \cdots + D_Nx^{(N)} &= \mathbf{b}_0 && \textit{(linking (global) constraints)} \\ A_ix^{(i)} &= \mathbf{b}_i && \forall i \in [N] \textit{ (local constraints)} \\ \mathbf{l}_i \leq x^{(i)} &\leq \mathbf{u}_i && \forall i \in [N] \textit{ (box constraints)} \end{aligned}$$

where we have $D_i \in \mathbb{Z}^{r \times t_i}$, $A_i \in \mathbb{Z}^{s_i \times t_i}$, $\mathbf{b}_0 \in \mathbb{Z}^r$, $\mathbf{b}_i \in \mathbb{Z}^{s_i}$ and $\mathbf{l}_i, \mathbf{u}_i \in \mathbb{Z}^{t_i}$ for every $i \in [N]$. Let us denote $s = \max_{i \in [N]} s_i$ and recall that the dimension is $d = \sum_{i \in [N]} t_i$. There are r global constraints and, apart from these, each variable is only involved in at most s local constraints and one box constraint. We call variables not appearing in linking constraints *local*.

The current best algorithm solving the N-fold IP in $(rs\Delta)^{O(r^2s+rs^2)} d \log d \log D \log val_{\max}$ time is by Eisenbrand et al. [28, Cor. 97]; see also [27, 41], where D is the maximum range of a variable, $\Delta = \max_{i \in [N]} \Big(\max \big(\|D_i\|_\infty, \|A_i\|_\infty \big) \Big)$, and $val_{\max}$ is the maximum feasible value of the objective.

Variants of the Problem and Basic Observations. Let us formally introduce the considered problems and establish notation used in the next sections.

Directed Traveling Salesman Problem (DTSP)

Input:	A directed graph $G = (V, E)$, edge weights $\omega \colon E \to \mathbb{N}$, and a budget $b \in \mathbb{N}$.
Question:	Is there a closed directed walk in G of total weight at most b that traverses each vertex of G at least once?

Directed Waypoint Routing Problem (DWRP)

Input:	A directed graph $G = (V, E)$, set of waypoints $W \subseteq V$, edge weights $\omega \colon E \to \mathbb{N}$, edge capacities $\kappa \colon E \to \mathbb{N}$, budget $b \in \mathbb{N}$.
Question:	Is there a closed directed walk C in G of total weight at most b, that traverses each vertex in W (at least once) and such that for each edge e the number of times C traverses e is at most $\kappa(e)$?

To avoid degenerate cases, we assume that $|W|$ is always at least 2 and that the input graph is strongly connected. We use undirected parameters, and, when speaking about a width of a directed graph, we mean the width of the underlying undirected graph.

▶ **Lemma 6** (Folklore, see, e.g., Bang-Jansen and Gutin [7, Exercise 1.12]). *Any closed directed walk (in particular a solution to the problem) can be decomposed into simple cycles, such that the number of times an edge is traversed by the walk equals the number of the cycles it appears in.*

▶ **Observation 7** (⋆). *In each optimal solution to DWRP, each vertex v is visited at most $|W \setminus \{v\}|$ times.*

Hence, an instance of DsTSP can be interpreted as an equivalent instance of DWRP, by letting $\kappa(e) = |V|$ for every arc.

3 DWRP is FPT with Respect to the Feedback Edge Number

In this section we prove Theorem 2 that we restate here.

▶ **Theorem 2.** *DWRP can be solved in $k^{O(k)} + n^{O(1)}$ time, where k is the feedback edge number of the input graph.*

Proof. Let $(G = (V, E), W, \omega, \kappa, b)$ be an instance of DWRP, where $n = |V|$. Let G' be the underlying undirected graph of G. Let $F \subseteq E(G')$ be a feedback edge set of G' of size at most k, i.e., $G' \setminus F$ is a forest. Note that it can be computed in polynomial time.

Our algorithm consists of three phases. First, we exhaustively apply some straightforward reduction rules in order to simplify (or already reject) the input. Then, we branch into $2^{O(k)}$ of possibilities, guessing the structure of the solution. Finally, for each such guess, we create an equivalent instance of DWRP with $O(k)$ vertices and edges. The original instance is a yes-instance if and only if at least one of the created instances is a yes-instance. Thus, our algorithm can be seen as a compression to OR of $2^{O(k)}$ instances, each of size $O(k)$. By Observation 7, each such instance can be solved in time $k^{O(k)}$ by brute force. Thus, the overall running time is $k^{O(k)} + n^{O(1)}$.

Preprocessing. The following reduction rule lets us deal with vertices of degree 1 in G'.

▶ **Reduction Rule 1.** *Suppose there is a vertex v in G which has only one neighbor u in G'.*

- *If $v \notin W$, remove v from the graph.*
- *If $v \in W$ and both $(u, v) \in E$ and $(v, u) \in E$, then remove the vertex v, include u into W, and decrease the budget by $\omega((u, v)) + \omega((v, u))$.*
- *If $v \in W$ and one of the arcs $(u, v), (v, u)$ is not present, then reject the instance.*

The correctness of the rule is straightforward. Note that, as $\{u, v\}$ is not part of any cycle in G', we may assume that $\{u, v\} \notin F$, and thus F is still a feedback edge set of the graph resulting from the reduction rule.

We apply the reduction rule exhaustively, and, for simplicity, we keep denoting the resulting instance by $(G, W, \omega, \kappa, b)$ and the underlying undirected graph by G'.

Let R be the set of all vertices that are incident to the edges in F. Note that since the reduction rule was exhaustively applied, each vertex has degree at least two in G'. Therefore, each leaf of $G'' = G' \setminus F$ is in R and, thus, G'' has at most $2k$ leaves. Let D be the set of vertices that have degree at least 3 in G''. As G'' is a forest with at most $2k$ leaves, the size of D is at most $2k - 1$. We also define $X = R \cup D$, and we have $|X| \leq 4k - 1$.

Let $\mathcal{P}'$ contain all paths in G'' with both endpoints in X and all internal vertices outside X. In other words, the paths in $\mathcal{P}'$ correspond to the edges of the forest obtained from G'' by contracting all vertices of degree 2. As there are at most $4k - 1$ vertices in X and, as G'' is a forest, the number of paths in $\mathcal{P}'$ is at most $4k - 2$. Finally, we define $\mathcal{P} = \mathcal{P}' \cup F$; we clearly have $|\mathcal{P}| \leq 5k - 2$.

Types of paths and their costs. Consider an optimum solution C^*, i.e., a shortest (in terms of the sum of weights) closed walk that visits every vertex from W in G. Among all such solutions, pick one with the minimum number of edges.

Consider a path $P \in \mathcal{P}$ with endpoints $u, v \in X$. A *visit* is an inclusion-wise maximal set of orientations of edges from P that appear consecutively in C^*. A visit is a *pass* if it contains for every edge of P exactly one of its orientations and it contains it exactly once, i.e., it corresponds to traversing P once, either from u to v or from v to u.

Notice that a path P might be of one of the following types:

($u \curvearrowright v$) there is at least one visit and every visit is a pass from u to v,

($u \curvearrowleft v$) there is at least one visit and every visit is a pass from v to u,

($u \leftrightarrows v$) every visit is a pass and there is at least one pass in each direction,

($u \not\leftrightarrow v$) none of the visits is a pass, there is a visit that starts in u and ends in u, but no visit that starts in v and ends in v,

$(u \twoheadrightarrow v)$ none of the visits is a pass, there is a visit that starts in v and ends in v, but no visit that starts in u and ends in u,

$(u \circlearrowleft v)$ none of the visits is a pass, there is a visit that starts in u and ends in u, and a visit that that starts in v and ends in v,

$(u \cdots v)$ the path P is not visited at all by C^*.

▷ Claim 8 ($\star$). The types listed above are mutually exclusive and cover all possibilities.

Let us make some further observations about possible visits in P by C^*.

If no internal vertex of P is in W, then P is of type $u \curvearrowleft v$, $u \curvearrowright v$, $u \leftrightarrows v$, or $u \cdots v$. Indeed, any non-pass visit can be removed from C^*, contradicting the minimality.

Now consider a path P of type $u \twoheadleftarrow v$ (the case $u \twoheadrightarrow v$ is symmetric). We may safely assume that P is visited exactly once by C^*, as one of the visits (the one reaching furthest towards v) covers all vertices covered by all the visits altogether. Further, we may assume that this single visit starts in u, reaches the internal vertex of P that belongs to W and is closest to v, and then goes back to u. Thus, if P is of type $u \twoheadleftarrow v$ or $u \twoheadrightarrow v$, then the visit in P contributes to the total cost of the solution by a fixed amount that can be computed in advance. We denote this cost by $\mathsf{cost}(P, u \twoheadleftarrow v)$ or $\mathsf{cost}(P, v \twoheadrightarrow u)$, respectively.

Now consider a path P of type $u \circlearrowleft v$. Similarly as before, we can assume that there are at least two vertices from W among internal vertices of P, as otherwise we can shorten C^* by removing one of the visits. Furthermore, we can assume that P is visited exactly twice: one visit starts at, u reaches some vertex $x_u \in W$ and goes back to u, and the other visit starts at v, reaches some vertex $x_v \in W$, and goes back to v. In particular, no edge between x_u and x_v is traversed by C^*. As before, in polynomial time we can find an optimal pair of visits that cover all vertices from W that are on P, and compute its contribution to the cost of the solution (here note that we need to be careful as some edges might be only available in one direction). We denote this cost by $\mathsf{cost}(P, u \circlearrowleft v)$.

Finally, each pass in a fixed direction has a fixed cost, equal to the sum of weights of edges in G corresponding to the edges of P, where we only choose edges in the correct direction. We denote it by $\mathsf{cost}(P, u \curvearrowright v)$ or $\mathsf{cost}(P, v \curvearrowright u)$, depending on the direction. We also define $\mathsf{capacity}(P, u \curvearrowright v)$ (and, symmetrically, $\mathsf{capacity}(P, u \curvearrowleft v)$) to the minimum capacity of an edge of G corresponding to an edge of P, in the direction from u to v (from v to u), resp.

Note that due to the directions and capacities of edges, some types might not be available for P. We will define the cost associated with such an unavailable type to be ∞.

Building compressed instances. Now we perform the branching. For each path P in $\mathcal{P}$ with endvertices u, v, we guess its type $u \sim v$, where $\sim \in \{\curvearrowleft, \curvearrowright, \leftrightarrows, \twoheadleftarrow, \circlearrowleft, \twoheadrightarrow, \cdots\}$. This results in $7^{|\mathcal{P}|} = 2^{O(k)}$ branches.

Consider one such branch. Now we build an instance $(G', W', \omega', \kappa', b')$ of DWRP. Initialize $b' = b$. We start with including all vertices from X to G', and all vertices from $X \cap W$ to W'. Now let P be a path in $\mathcal{P}$ with endvertices $u, v \in X$. We proceed, depending on $\sim$. If the type of P is $u \cdots v$, we do nothing. If the type of P is $u \twoheadleftarrow v$ ($\twoheadrightarrow$), we add u (v) to W', and decrease b' by $\mathsf{cost}(P, u \twoheadleftarrow v)$ ($\mathsf{cost}(P, v \twoheadrightarrow u)$), respectively. If the type of P is $u \circlearrowleft v$, we add both u and v to W', and decrease b' by $\mathsf{cost}(P, u \circlearrowleft v)$.

Now consider the case that the type of P is $u \curvearrowright v$ (the case $u \curvearrowleft v$ is symmetric). We add a new vertex $x^P_{u \curvearrowright v}$ and edges $(u, x^P_{u \curvearrowright v})$ and $(x^P_{u \curvearrowright v}, v)$. Vertices u, v, and $x^P_{u \curvearrowright v}$ are all included into W'. We define $\omega'((u, x^P_{u \curvearrowright v})) = \mathsf{cost}(P, u \curvearrowright v)$ and $\omega'((x^P_{u \curvearrowright v}, v)) = 0$. Furthermore, we set $\kappa'((u, x^P_{u \curvearrowright v})) = \kappa'((x^P_{u \curvearrowright v}, v)) = \mathsf{capacity}(P, u \curvearrowright v)$. If the type of P is $u \leftrightarrows$, we proceed as if was of both types $u \curvearrowright v$ and $u \curvearrowleft v$.

This concludes the construction of the instance. If $b' < 0$, we reject it immediately. Clearly, the number of vertices of G' is at most $|X| + 2|\mathcal{P}| \leq 14k$, and the number of edges is at most $4|\mathcal{P}| \leq 20k$. The discussion about correctness is postponed to the full version. ◀

4 DWRP is FPT Parameterized by Vertex Integrity

In this section we prove Theorem 3.

▶ **Theorem 3.** *DWRP can be solved in* $k^{O(k^6)} \cdot n \cdot \log^2 n(\log n + \log U)$ *time, where* k *is the vertex integrity of the input graph and* U *is the maximum weight of an edge in* G*.*

Proof. An undirected graph G with vertex integrity k has *a k-modulator* $M \subseteq V(G)$ of size at most k such that each connected component of $G - M$ is of size at most k. We assume that G is given along with M as it can be found in $k^{O(k)}n$ time [26, 29]. Thus we assume that we are given an instance $(G, W, \omega, \kappa, b)$ of DWRP, with a set M of size at most k, such that each weakly connected component of $G - M$ has size at most k. For brevity, we will speak about "components" instead of "weakly connected components."

In what follows, we start with introducing some notions and establishing some properties of an optimum solution. Then, after some branching, we build an instance of ILP that corresponds to a solution with the required properties.

Segments. A *segment* is a walk in G with at least two vertices, that starts and ends in a vertex of M and all internal vertices are not in M. We remark that there might be no internal vertices – such a segment is just an arc contained in M. We call such a segment *trivial*. However, if a segment is non-trivial (i.e., not trivial), all its internal vertices are contained in a single component of $G - M$; the segment is *associated* with the component.

Note that the endpoints of a non-trivial segment need not to be distinct. A segment s from u to v is *minimal* if there is no segment from u to v that uses only the arcs of s, contains all vertices of s that are in W, and is shorter than s.

Segments in an Optimum Solution. Let C^* be an optimum solution, i.e., a shortest (w.r.t. the sum of weights of edges) closed directed walk in G that visits every vertex of W. We treat C^* as a sequence of vertices. Clearly C^* can be decomposed into segments (where two consecutive segments overlap on corresponding endpoints); let $\mathcal{S}^*$ be the collection of these segments. We emphasize that $\mathcal{S}^*$ is a multiset, as some segments might be used more than once. We observe that we can assume that each segment in $\mathcal{S}^*$ is minimal, for otherwise we can replace it with a shorter one with the same endpoints, still covering all of W.

Some segments in $\mathcal{S}^*$ might not be needed to cover W, but are required to keep the connectivity of the solution. We call them *connectors*. We can identify connectors with the following procedure. We initialize $\mathcal{S}^\circ = \emptyset$ and proceed iteratively as follows. If there is a segment s in $\mathcal{S}^* \setminus \mathcal{S}^\circ$ such that the segments $\mathcal{S}^* \setminus \mathcal{S}^\circ \setminus \{s\}$ cover all vertices of W, we include s into $\mathcal{S}^\circ$. We emphasize that here we work with multisets, so the "$\setminus$" operation respects the multiplicity of elements. The set $\mathcal{S}^\circ$ contains connectors. Note that $\mathcal{S}^\circ$ is not defined uniquely and it might depend on the ordering in which segments were selected.

Consider a connector, i.e., a segment s from u to v that is in $\mathcal{S}^\circ$. We observe that we can safely assume that it is a u-v-path, as otherwise we can replace it with such a path obtaining a solution of no larger length. We call such segments *simple*.

Traversals. Now let us consider a component C of $G - M$. A *traversal* of C is a collection $\mathcal{S}_C$ of segments associated with C that:

- together cover every vertex from $W \cap V(C)$,
- for every $s \in \mathcal{S}_C$ there is a vertex in $W \cap V(C)$ not covered by segments in $\mathcal{S}_C \setminus \{s\}$,
- every segment in $\mathcal{S}_C$ is minimal,
- every edge e is used at most $\kappa(e)$ times by $\mathcal{S}_C$.

Note that the segments from $\mathcal{S}^* \setminus \mathcal{S}^\circ$ associated with C form a traversal. Indeed, the first condition follows since C^* covers all vertices from W and we moved segments to $\mathcal{S}^\circ$ if they were not required to cover $W \cap V(C)$. The second condition again follows from the definition of $\mathcal{S}^\circ$. Finally, recall that we already established the third condition without loss of generality.

▷ Claim 9 (⋆). Every traversal has at most $k(k+2)$ vertices. Moreover, for any component C of $G - M$, there are at most $k^{k(k+2)}$ traversals of C. Furthermore, each traversal uses each edge at most k times.

Skeleton of the Solution. Let us define two directed graphs H_0 and H_1 as follows. Both have the same vertex set, M.

For distinct $u, v \in M$, the arc uv exists in H_0 if such an arc appears in C^* (i.e., C^* contains u immediately followed by v). On the other hand, the arc uv exists in H_1 if there is a non-trivial segment in $\mathcal{S}^*$ starting in u and ending in v. We emphasize that neither H_0 nor H_1 has parallel arcs nor loops.

Let H be the directed graph with vertex set M and the arc set being the union of arc sets of H_0 and H_1 (again, with no parallel arcs). Note that H has one strongly connected component H' containing all vertices from $W \cap M$, and vertices that are not in H' are isolated in H. These vertices do not appear on C^* at all.

For $u, v \in M$, we say that a segment from u to v is *bad* (for a particular choice of H_0, H_1) if u or v are not in H', or the arc uv does not appear in H. A traversal is *bad* if it contains a bad segment.

We say that a collection $\mathcal{S}$ of segments is compatible with H_0 and H_1 if, for every arc uv of H_0, $\mathcal{S}$ contains at least one copy of the trivial segment uv, and for every arc uv of H_1, $\mathcal{S}$ contains at least one non-trivial segment starting in u and ending in v.

The Algorithm. We proceed to the description of the algorithm. We exhaustively guess the graphs H_0 and H_1 described above, considering all possible graphs on the vertex set M, so that the union of these two graphs has one strongly connected component and the remaining vertices isolated. This gives us $2^{O(k^2)}$ possible guesses. We are going to look for a set of segments that are compatible with H_0 and H_1, and the structure of H_0, H_1 will be used to ensure that the solution is connected.

Suppose we are in a branch corresponding to one guess. We will build an instance of generalized N-fold ILP [27, 41]. In our case, we have a brick corresponding to each component C of $G - M$ and some further bricks that do not have any local constraints.

For each traversal of C that is not bad, we introduce one binary variable; recall that there are at most $k^{k(k+2)}$ such variables. The value of this variable indicates whether the vertices from $W \cap V(C)$ are covered by a particular traversal. Furthermore, we introduce a local constraint to ensure that exactly one traversal is chosen.

For each simple segment associated with C which is a u-v-path for some $u, v \in M$, we introduce a variable with possible values from 0 to n. The intended meaning of this variable is the number of times this segment was used as a connector from u to v by the solution. There are at most k^{k+2} such variables.

For each edge e with at least one endpoint in C we introduce a (local) constraint ensuring that this edge is used by the solution at most as many times as the capacity permits. Namely, we sum over all simple segments and over all traversals that use edge e (multiplying by how many times they use e) and require that this sum is at most $\kappa(e)$.

Now, let us move to variables not related to components. For each ordered pair u, v of distinct vertices, we introduce a variable with possible values from 0 to $\kappa((u, v))$. The intended meaning of this variable is the number of times the trivial segment is used as a connector from u to v by the solution. This gives us at most k^2 variables. We can put each of this variable to a brick for itself. As the edges within M are not used by any traversals or non-trivial segments, the box constraints ensure that the capacities are adhered for these arcs, i.e., there will be no local constraints for these bricks.

Next, we add global constraints that ensure that the selected set of segments is compatible with H_0 and H_1. More precisely, for each arc uv of H_0 (H_1), we add a constraint that ensures that we choose the trivial segment uv (at least one non-trivial segment that starts in u and ends in v), respectively. We sum it over all selected traversals and connectors. These constraints are responsible for making the solution connected.

Finally, we need to make sure that for each $v \in M$, the number of segments that start in v is equal to the number of segments that end in v. Again, we can force it by summing the number of particular segments, including trivial, connectors, and ones belonging to traversals. These are also global constraints.

We minimize the total length (i.e., the sum of weights of edges) in all selected segments. Note that for some guesses we might get an infeasible instance of ILP – this means that, e.g., the segments that are required by H_0 and H_1 do not exist in G or all traversals of some component are bad. If all branches give infeasible instances, we return that there is no solution. However, every solution found (in any branch) is feasible and we can return the shortest one. From the description above it follows that if an optimum solution C^* exist, we will find it (or, more precisely, a solution of the same total length) in the branch corresponding to the actual H_0 and H_1.

Solving N-fold ILP. Recall that generalized N-fold ILP can be solved (see [28, Cor. 97] and also [27, 41]) in $(rs\Delta)^{O(r^2s+rs^2)} d \log d \log D \log val_{\max}$ time, where

- r is the number of global constraints (at most $k(k-1)+k = k^2$ in our case),
- s is the number of local constraints for each brick ($O(k^2)$ in our case),
- Δ is the largest coefficient in the constraints (k in our case),
- d is the total number of variables (at most $k^2 + n \cdot k^{k(k+2)} + n \cdot k^{k+2}$ in our case)
- D is the maximum range of a variable (at most n in our case),
- and $val_{\max}$ is the maximum value of the objective function (by Observation 7, in our case it is at most $n^2 \cdot U$, where U is the maximum weight of an arc in G).

Including the branching to guess H_0 and H_1, we obtain the final running time bound of $k^{O(k^6)} \cdot n \cdot \log^2 n(\log n + \log U)$. This completes the proof. ◀

5 W[1]-hardness with Respect to Distance to Constant Treedepth

In this section we prove Theorem 5.

▶ **Theorem 5.** *DWRP is W[1]-hard with respect to modulator to constant treedepth, even if all weights are 1.*

We provide a reduction from Capacitated Dominating Set (CDS), which is defined as follows. We are given an undirected graph G and a *capacity function* $c\colon V(G) \to \mathbb{N}$, where $1 \leq c(u) \leq \deg_G(u)$ for every $u \in V(G)$. A set of vertices $S \subseteq V(G)$ is a *capacitated dominating set* if there exists a *domination mapping* $f\colon (V(G) \setminus S) \to S$ such that for every $v \in (V(G) \setminus S)$ we have $\{f(v), v\} \in E(G)$ and each vertex $s \in S$ dominates at most $c(s)$ vertices, i.e., $|f^{-1}(s)| \leq c(s)$. Given a $k \in \mathbb{N}$, the problem is to find a capacitated dominating set of size k. CDS was shown to be W[1]-hard with respect to treewidth in [25]. In order to prove Theorem 5, we actually need to observe a slight improvement of the construction proposed in [25]: we require that CDS is W[1]-hard with respect to the distance to constant treedepth. This W[1]-hardness is obtained by carefully considering the structure of the graph used in the reduction in [25]. Formally:

▶ **Theorem 10** ($\star$). *CDS is W[1]-hard with respect to the distance to stars.*

We are now ready to proceed with the sketch of the proof of Theorem 5. A more detailed version can be found in the full version.

Sketch of Proof of Theorem 5. Let $I = (G = (V, E), c, k)$ be an instance of CDS. We describe the construction of an instance $I' = (G', W, \omega, \kappa, b)$ of DWRP that is equivalent to I such that G' has a bounded modulator to constant treedepth.

The construction. First, we describe the construction of the used gadgets. Each gadget has dedicated *connecting vertices* marked with "in" (e.g. u_{in}), "out", or "io". Every gadget of the final G' will be connected to other gadgets by edges incident only to its connecting vertices: "in" ("out" resp.) vertices can only be incident to arcs that are coming "into" (going "out" from resp.) the gadget, and "io" can be incident to both types of arcs. If this is true then we say that the graph *contains* the gadget. We use u^X to denote the vertex u that belongs to a gadget X. All edges have weight 1. Unless stated otherwise, every edge has capacity n; this is always sufficient by Observation 7. Crucially, getting from one terminal to another requires the traversal of exactly two non-terminal vertices. As the final budget is set to exactly $3 \cdot$ (number of terminals), it follows that every terminal is visited exactly once.

We begin with the *force-p-traversals-gadget*. It consists of three non-terminal vertices $u_{\text{in}}, u_{\text{out}}, w$ and terminal vertices $v_1, \ldots, v_p$ joined with edges $(u_{\text{in}}, v_i), (v_i, w)$ for every $i \in [p]$ and (w, u_{out}) of capacity p, see Figure 2.

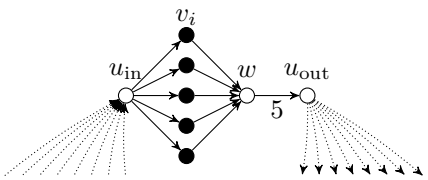


Figure 2 Force-5-traversals-gadget. Black and white vertices represent terminals and non-terminals, respectively.

▷ Claim 11. Let G' be a graph that contains a force-p-traversals-gadget X. Then every solution to DWRP in G' traverses from u^X_{in} to u^X_{out} exactly p times.

Next, we need the *cover-z-gadget*, which is illustrated in Figure 3. Let R_1 be the directed path that starts at s_{in}, goes to f, then to b, continues to x_{in}, traverses all the ws' until y_{out}, goes to c and finishes in t_{out}. Also, let R_2 and P be the red and blue paths depicted

in Figure 3. The idea here is that this gadget has two possible behaviors: either its terminals are covered by the combination of the R_2 and P paths, or all of its terminals except from z_{io} are covered by the R_1 path, and z is covered by some external path that visits this gadget. Formally:

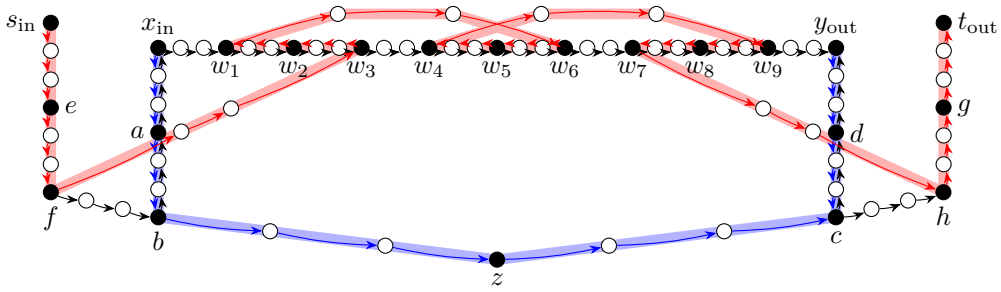


Figure 3 The cover-z-gadget. Black and white vertices represent terminals and non-terminals, respectively. The path R_2 is colored red, while the path P is colored blue.

▷ Claim 12. Let G be a graph that contains the cover-z-gadget X. If S is a minimum solution to DWRP that traverses every terminal vertex of X exactly once, then S includes either the path R_1, or the paths P and R_2 as continuous subpaths.

We now present the construction of $(G', W, \omega, \kappa, b)$, see Figure 4. Let $\mathcal{S}$ be a force-$(k+1)$-traversals-gadget that we call *selection-gadget* and let $\mathcal{A}$ be a force-$(2|E|+|V|+1)$-traversals-gadget. We create an auxiliary non-terminal vertex x and add an arc $(x, u^{\mathcal{A}}_{\text{in}})$. Let x together with $\mathcal{A}$ be called the *auxiliary-gadget*. We connect the selection- and auxiliary-gadgets by creating terminal vertices x_1, x_2, non-terminal vertex x_3, and arcs $(u^{\mathcal{A}}_{\text{out}}, x_2)$, (x_2, x_3), $(x_3, u^{\mathcal{S}}_{\text{in}})$, $(u^{\mathcal{S}}_{\text{out}}, x_1)$, (x_1, x). These two gadgets compose the *upper part* of G'.

Finally, we encode the vertices and edges of G into $G' = (V', E')$. For every vertex $u \in V$ we add vertices z_u, c_u, and d_u to V', then we create a cover-z_u-gadget denoted $\mathcal{E}^u$. We add arcs $(u^{\mathcal{S}}_{\text{out}}, x^{\mathcal{E}^u}_{\text{in}})$, $(y^{\mathcal{E}^u}_{\text{out}}, c_u)$, (c_u, d_u), $(c_u, u^{\mathcal{S}}_{\text{in}})$, $(u^{\mathcal{A}}_{\text{out}}, s^{\mathcal{E}^u}_{\text{in}})$, $(t^{\mathcal{E}^u}_{\text{out}}, x)$ to E', and we set the capacity of (c_u, d_u) to $c(u)$. For every arc (u, v) such that $\{u, v\} \in E$, we create a cover-z_v-gadget denoted $\mathcal{E}^{u,v}$. Then we add arcs $(d_u, x^{\mathcal{E}^{u,v}}_{\text{in}})$, $(y^{\mathcal{E}^{u,v}}_{\text{out}}, c_u)$, $(u^{\mathcal{A}}_{\text{out}}, s^{\mathcal{E}^{u,v}}_{\text{in}})$, $(t^{\mathcal{E}^{u,v}}_{\text{out}}, x)$. Note that in the above we created two gadgets, $\mathcal{E}^{u,v}$ and $\mathcal{E}^{v,u}$, for each $\{u, v\} \in E$. Finally, we set the budget $b = 132|E| + 69|V| + 3k + 12$. This concludes construction of the DWRP instance.

The instances are equivalent. On a high level, we show that any solution C of the instance $(G', W, \omega, \kappa, b)$ obeying the above budget has a specific behavior. First, the budget is set so that each terminal can be visited at most once. Let $S \subseteq V$ be a solution of I. The walk C is separated into two stages. In the first stage, for each $u \in S$ the walk C starts from $u^{\mathcal{S}}_{\text{in}}$, goes through $\mathcal{S}$ to $u^{\mathcal{S}}_{\text{out}}$, from there to $x^{\mathcal{E}^u}_{\text{in}}$, using the P path (visiting z_u) of the gadget $\mathcal{E}^u$ to $y^{\mathcal{E}^u}_{\text{out}}$ and to c_u. Then for each vertex v that is dominated by u, it goes from c_u to d_u, uses the P path of gadget $\mathcal{E}^{u,v}$ to visit z_v and return to c_u. Crucially, this can be done at most $c(u)$ times. Finally it returns to $u^{\mathcal{S}}_{\text{in}}$.

Once all the vertices of S are dealt with in this way the first stage is done, and C proceeds to $\mathcal{A}$. For each cover-gadget $\mathcal{E}$ (which are equal in number to the terminals of $\mathcal{A}$ minus one), the walk C goes from x through $\mathcal{A}$ to $u^{\mathcal{A}}_{\text{out}}$ and then through $\mathcal{E}$ to cover the "remaining" vertices of $\mathcal{E}$ (i.e., the vertices that were not visited during the first stage); this is done by

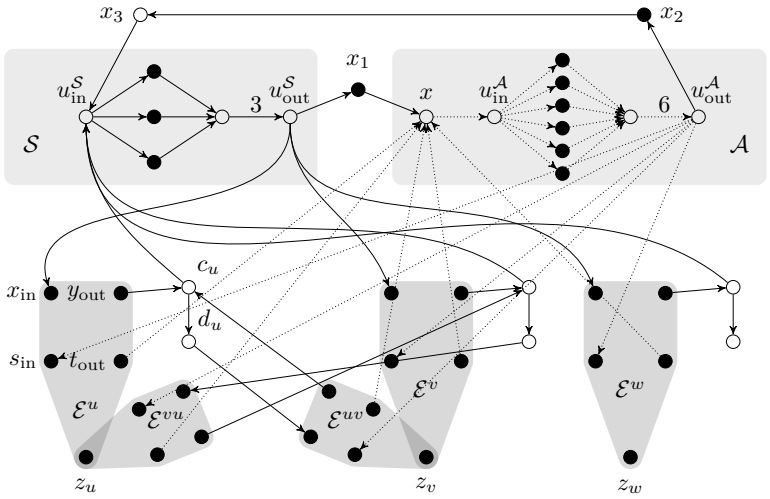


Figure 4 An example of a completely constructed DWRP instance for graph $G = (\{u, v, w\}, \{\{u, v\}\})$ and $k = 2$ (capacities omitted). The top half contains the upper part of G'. The bottom half corresponds to the vertices and edges of G. Gray boxes hide the inner parts of the corresponding cover-gadgets, see Figure 3. Dotted arrows signify arcs that belong to the auxiliary-gadget or connect cover-gadgets to the auxiliary-gadget. Black and white vertices represent terminals and non-terminals, respectively.

employing either the R_1 or the R_2 path, according to whether $\mathcal{E}$ was traversed during the first step. We then show that C exists and is of the correct budget if and only if I is a yes-instance of CDS.

The rest of the proof focuses on proving that G' has a modulator of size $7|M| + 7$ to treedepth 86, using that $G' \setminus M$ has treedepth 2 (it is a disjoint union of stars). ◀

6 Conclusions and Open Problems

We have shown that DWRP is FPT w.r.t. vertex integrity as well as w.r.t. feedback edge set number, while it is W[1]-hard w.r.t. distance to constant treedepth even in unweighted graphs. DTSP is W[1]-hard w.r.t. distance to pathwidth 3, also even in unweighted graphs and we conjecture that the above hardness w.r.t. distance to constant treedepth holds also for this problem. An immediate open problem stemming from our research is to close the gap by considering distances to even smaller constant values of the parameters. Namely, we would like to know whether the problems are FPT w.r.t. feedback vertex set (distance to treewidth 1), distance to disjoint paths (pathwidth 1), or distance to stars (treedepth 2).

We have also shown, that the problems are FPT parameterized by treewidth combined with the number of visits. Given that all the hardness reductions leverage vertices of high degrees, we are wondering whether a combination of treewidth with the maximum degree of the input graph might lead to an fpt-algorithm.

In this paper, we focused on structural parameters of the underlying undirected graph. Nevertheless, there are many width measure actually designated for directed graphs [31] and the tractability w.r.t. them is open. In fact, currently we do not even know whether DTSP is polynomial time solvable or NP-hard on directed graphs that become acyclic (DAGs) after deleting a single vertex or even a single edge.

References

1 Saeed Akhoondian Amiri, Klaus-Tycho Foerster, Riko Jacob, Mahmoud Parham, and Stefan Schmid. Waypoint routing in special networks. In Claudio Casetti, Fernando A. Kuipers, James P. G. Sterbenz, and Burkhard Stiller, editors, *2018 IFIP Networking Conference and Workshops, Networking 2018*, pages 496–504. IFIP, 2018. doi:10.23919/IFIPNetworking.2018.8696560.

2 Saeed Akhoondian Amiri, Klaus-Tycho Foerster, Riko Jacob, and Stefan Schmid. Charting the algorithmic complexity of waypoint routing. *Comput. Commun. Rev.*, 48(1):42–48, 2018. doi:10.1145/3211852.3211859.

3 Saeed Akhoondian Amiri, Klaus-Tycho Foerster, and Stefan Schmid. Walking through waypoints. *Algorithmica*, 82(7):1784–1812, 2020. doi:10.1007/s00453-020-00672-z.

4 Nima Anari and Shayan Oveis Gharan. Effective-resistance-reducing flows, spectrally thin trees, and asymmetric TSP. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015*, pages 20–39. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.11.

5 David L. Applegate, Robert E. Bixby, Vašek Chvátal, and William J. Cook. *The Traveling Salesman Problem: A Computational Study*, volume 17 of *Princeton Series in Applied Mathematics*. Princeton University Press, Princeton, 2011. doi:10.1515/9781400841103.

6 Arash Asadpour, Michel X. Goemans, Aleksander Madry, Shayan Oveis Gharan, and Amin Saberi. An $O(\log n/\log\log n)$-approximation algorithm for the asymmetric traveling salesman problem. *Oper. Res.*, 65(4):1043–1061, 2017. doi:10.1287/OPRE.2017.1603.

7 Jørgen Bang-Jensen and Gregory Z. Gutin. *Digraphs – Theory, Algorithms and Applications, Second Edition*. Springer Monographs in Mathematics. Springer, 2009.

8 Lukas Behrendt, Katrin Casel, Tobias Friedrich, J. A. Gregor Lagodzinski, Alexander Löser, and Marcus Wilhelm. From symmetry to asymmetry: Generalizing TSP approximations by parametrization. *J. Comput. Syst. Sci.*, 136:157–170, 2023. doi:10.1016/J.JCSS.2023.03.007.

9 Richard Bellman. Dynamic programming treatment of the travelling salesman problem. *Journal of the ACM*, 9(1):61–63, 1962. doi:10.1145/321105.321111.

10 Mark de Berg, Kevin Buchin, Bart M. P. Jansen, and Gerhard Woeginger. Fine-grained complexity analysis of two classic TSP variants. *ACM Transactions on Algorithms*, 17(1):5:1–5:29, 2021. doi:10.1145/3414845.

11 André Berger, László Kozma, Matthias Mnich, and Roland Vincze. Time- and space-optimal algorithm for the many-visits TSP. *ACM Trans. Algorithms*, 16(3):35:1–35:22, 2020. doi:10.1145/3382038.

12 René van Bevern and Daniel A. Skachkov. A quadratic-order problem kernel for the traveling salesman problem parameterized by the vertex cover number. *Operations Research Letters*, 52:107065, 2024. doi:10.1016/j.orl.2024.107065.

13 Václav Blažej, Pratibha Choudhary, Dušan Knop, Šimon Schierreich, Ondřej Suchý, and Tomáš Valla. On polynomial kernels for traveling salesperson problem and its generalizations. In Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman, editors, *30th Annual European Symposium on Algorithms, ESA 2022*, volume 244 of *LIPIcs*, pages 22:1–22:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ESA.2022.22.

14 Václav Blažej, Andreas Emil Feldmann, Foivos Fioravantes, Pawel Rzazewski, and Ondřej Suchý. Parameterized complexity of directed traveling salesman problem. *CoRR*, abs/2506.22127, 2025. doi:10.48550/arXiv.2506.22127.

15 Hans-Joachim Böckenhauer, Juraj Hromkovic, Joachim Kneis, and Joachim Kupke. The parameterized approximability of TSP with deadlines. *Theory Comput. Syst.*, 41(3):431–444, 2007. doi:10.1007/S00224-007-1347-X.

16 Édouard Bonnet, Yoichi Iwata, Bart M. P. Jansen, and Łukasz Kowalik. Fine-grained complexity of k-OPT in bounded-degree graphs for solving TSP. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *Proceedings of the 27th Annual European Symposium on Algorithms, ESA '19*, volume 144 of *Leibniz International Proceedings in Informatics*, pages 23:1–23:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ESA.2019.23.

17 Édouard Bonnet, Michael Lampis, and Vangelis Th. Paschos. Time-approximation trade-offs for inapproximable problems. *J. Comput. Syst. Sci.*, 92:171–180, 2018. doi:10.1016/J.JCSS.2017.09.009.

18 Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. *Operations Research Forum*, 3:20, 2022. doi:10.1007/s43069-021-00101-z.

19 W. Cook. *In Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation*. Princeton University Press, 2011. doi:10.1515/9781400839599.

20 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009. URL: http://mitpress.mit.edu/books/introduction-algorithms.

21 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer International Publishing, Cham, 2015. doi:10.1007/978-3-319-21275-3.

22 Marek Cygan, Łukasz Kowalik, and Arkadiusz Socała. Improving TSP tours using dynamic programming over tree decompositions. *ACM Transactions on Algorithms*, 15(4):54:1–54:19, 2019. doi:10.1145/3341730.

23 Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Fast hamiltonicity checking via bases of perfect matchings. *J. ACM*, 65(3):12:1–12:46, 2018. doi:10.1145/3148227.

24 Reinhard Diestel. *Graph Theory, 6th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2025.

25 Michael Dom, Daniel Lokshtanov, Saket Saurabh, and Yngve Villanger. Capacitated domination and covering: A parameterized perspective. In Martin Grohe and Rolf Niedermeier, editors, *Parameterized and Exact Computation, Third International Workshop, IWPEC 2008*, volume 5018 of *Lecture Notes in Computer Science*, pages 78–90. Springer, 2008. doi:10.1007/978-3-540-79723-4_9.

26 Pål Grønås Drange, Markus S. Dregi, and Pim van 't Hof. On the computational complexity of vertex integrity and component order connectivity. *Algorithmica*, 76(4):1181–1202, 2016. doi:10.1007/S00453-016-0127-X.

27 Friedrich Eisenbrand, Christoph Hunkenschröder, and Kim-Manuel Klein. Faster algorithms for integer programs with block structure. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP '18*, volume 107 of *Leibniz International Proceedings in Informatics*, pages 49:1–49:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.49.

28 Friedrich Eisenbrand, Christoph Hunkenschröder, Kim-Manuel Klein, Martin Koutecký, Asaf Levin, and Shmuel Onn. An algorithmic theory of integer programming. *CoRR*, abs/1904.01361, 2019. arXiv:1904.01361.

29 Michael R Fellows and Sam Stueckle. The immersion order, forbidden subgraphs and the complexity of network integrity. *J. Combin. Math. Combin. Comput*, 6(1):23–32, 1989.

30 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. Clique-width III: Hamiltonian cycle and the odd case of graph coloring. *ACM Trans. Algorithms*, 15(1), 2018. doi:10.1145/3280824.

31 Robert Ganian, Petr Hlinený, Joachim Kneis, Daniel Meister, Jan Obdrzálek, Peter Rossmanith, and Somnath Sikdar. Are there any good digraph width measures? *J. Comb. Theory B*, 116:250–286, 2016. doi:10.1016/J.JCTB.2015.09.001.

32 M. R. Garey, David S. Johnson, and Robert Endre Tarjan. The planar hamiltonian circuit problem is NP-complete. *SIAM J. Comput.*, 5(4):704–714, 1976. doi:10.1137/0205049.

33 Shayan Oveis Gharan, Amin Saberi, and Mohit Singh. A randomized rounding approach to the traveling salesman problem. In Rafail Ostrovsky, editor, *Proceedings of the 52nd IEEE Annual Symposium on Foundations of Computer Science, FOCS '11*, pages 550–559. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.80.

34 Jiong Guo, Sepp Hartung, Rolf Niedermeier, and Ondřej Suchý. The parameterized complexity of local search for TSP, more refined. *Algorithmica*, 67(1):89–110, 2013. doi:10.1007/s00453-012-9685-8.

35 Gregory Gutin and Abraham P. Punnen, editors. *The Traveling Salesman Problem and Its Variations*. Springer, 2007. doi:10.1007/b101971.

36 Michael Held and Richard M. Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210, 1962. doi:10.1137/0110015.

37 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.

38 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/JCSS.2001.1774.

39 Marek Karpinski, Michael Lampis, and Richard Schmied. New inapproximability bounds for TSP. *J. Comput. Syst. Sci.*, 81(8):1665–1677, 2015. doi:10.1016/J.JCSS.2015.06.003.

40 Philip N. Klein and Dániel Marx. A subexponential parameterized algorithm for subset TSP on planar graphs. In Chandra Chekuri, editor, *Proceedings of the 2014 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '14*, pages 1812–1830. SIAM, 2014. doi:10.1137/1.9781611973402.131.

41 Martin Koutecký, Asaf Levin, and Shmuel Onn. A parameterized strongly polynomial algorithm for block structured integer programs. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP '18*, volume 107 of *Leibniz International Proceedings in Informatics*, pages 85:1–85:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.85.

42 Lukasz Kowalik, Shaohua Li, Wojciech Nadara, Marcin Smulewicz, and Magnus Wahlström. Many-visits TSP revisited. *J. Comput. Syst. Sci.*, 124:112–128, 2022. doi:10.1016/J.JCSS.2021.09.007.

43 Michael Lampis. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica*, 64(1):19–37, 2012. doi:10.1007/S00453-011-9554-X.

44 Giuseppe Lancia and Marcello Dalpasso. Algorithmic strategies for a fast exploration of the TSP 4-opt neighborhood. *J. Heuristics*, 30(3-4):109–144, 2024. doi:10.1007/S10732-023-09523-W.

45 Isja Mannens, Jesper Nederlof, Céline M. F. Swennenhuis, and Krisztina Szilágyi. On the parameterized complexity of the connected flow and many visits TSP problem. In Lukasz Kowalik, Michal Pilipczuk, and Pawel Rzazewski, editors, *Graph-Theoretic Concepts in Computer Science - 47th International Workshop, WG 2021*, volume 12911 of *Lecture Notes in Computer Science*, pages 52–79. Springer, 2021. doi:10.1007/978-3-030-86838-3_5.

46 Dániel Marx. Searching the k-change neighborhood for TSP is W[1]-hard. *Operations Research Letters*, 36(1):31–36, 2008. doi:10.1016/j.orl.2007.02.008.

47 Dániel Marx, Marcin Pilipczuk, and Michal Pilipczuk. A subexponential parameterized algorithm for directed subset traveling salesman problem on planar graphs. *SIAM J. Comput.*, 51(2):254–289, 2022. doi:10.1137/19M1304088.

48 Dániel Marx, Ario Salmasi, and Anastasios Sidiropoulos. Constant-factor approximations for asymmetric TSP on nearly-embeddable graphs. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2016*, volume 60 of *LIPIcs*, pages 16:1–16:54. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICS.APPROX-RANDOM.2016.16.

49 Tobias Mömke and Ola Svensson. Removing and adding edges for the traveling salesman problem. *Journal of the ACM*, 63(1):2:1–2:28, 2016. doi:10.1145/2739008.

50 Marcin Mucha. 13/9-approximation for graphic TSP. *Theory of Computing Systems*, 55:640–657, 2014. doi:10.1007/s00224-012-9439-7.

51 Šimon Schierreich and Ondřej Suchý. Waypoint routing on bounded treewidth graphs. *Information Processing Letters*, 173:106165, 2022. doi:10.1016/j.ipl.2021.106165.

52 András Sebö 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.

53 D.B. Shmoys, J.K. Lenstra, A.H.G. Rinnooy Kan, and E.L. Lawler. *The Traveling Salesman Problem*, volume 12 of *Wiley Series in Discrete Mathematics & Optimization*. John Wiley & Sons, Incorporated, 1985. URL: https://www.wiley.com/en-ie/The+Traveling+Salesman+Problem%3A+A+Guided+Tour+of+Combinatorial+Optimization-p-9780471904137.

54 Ola Svensson. Approximating ATSP by relaxing connectivity. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015*, pages 1–19. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.10.

55 Ola Svensson, Jakub Tarnawski, and László A. Végh. A constant-factor approximation algorithm for the asymmetric traveling salesman problem. *J. ACM*, 67(6):37:1–37:53, 2020. doi:10.1145/3424306.

56 Vera Traub and Jens Vygen. An improved approximation algorithm for the asymmetric traveling salesman problem. *SIAM J. Comput.*, 51(1):139–173, 2022. doi:10.1137/20M1339313.

A Dichotomy for 1-Planarity with Restricted Crossing Types Parameterized by Treewidth

Sergio Cabello
Faculty of Mathematics and Physics, University of Ljubljana, Slovenia
Institute of Mathematics, Physics and Mechanics, Ljubljana, Slovenia

Alexander Dobler
TU Wien, Austria

Gašper Fijavž
Faculty of Computer and Information Science, University of Ljubljana, Slovenia

Thekla Hamm
Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands

Mirko H. Wagner
Theoretical Computer Science, Osnabrück University, Germany

Abstract

A drawing of a graph is 1-planar if each edge participates in at most one crossing and adjacent edges do not cross. Up to symmetry, each crossing in a 1-planar drawing belongs to one out of six possible crossing types, where a type characterizes the subgraph induced by the four vertices of the crossing edges. Each of the 63 possible nonempty subsets $\mathcal{S}$ of crossing types gives a recognition problem: does a given graph admit an $\mathcal{S}$-restricted drawing, that is, a 1-planar drawing where the crossing type of each crossing is in $\mathcal{S}$?

We show that there is a set $\mathcal{S}_{\text{bad}}$ with three crossing types and the following properties:

- If $\mathcal{S}$ contains no crossing type from $\mathcal{S}_{\text{bad}}$, then the recognition of graphs that admit an $\mathcal{S}$-restricted drawing is fixed-parameter tractable with respect to the treewidth of the input graph.
- If $\mathcal{S}$ contains any crossing type from $\mathcal{S}_{\text{bad}}$, then it is NP-hard to decide whether a graph has an $\mathcal{S}$-restricted drawing, even when considering graphs of constant pathwidth.

We also extend this characterization of crossing types to 1-planar straight-line drawings and show the same complexity behaviour parameterized by treewidth.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Fixed parameter tractability

Keywords and phrases 1-planar, crossing type, treewidth, pathwidth

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.16

Funding *Sergio Cabello*: Funded in part by the Slovenian Research and Innovation Agency (P1-0297, N1-0218, N1-0285). Funded in part by the European Union (ERC, KARST, project number 101071836). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.
Alexander Dobler: Vienna Science and Technology Fund (WWTF) grant [10.47379/ICT19035].

Acknowledgements This work was started during the Crossing Number Workshop 2024 that took place in Montserrat, Spain. We thank the organizers and participants for providing a fruitful research environment.

© Sergio Cabello, Alexander Dobler, Gašper Fijavž, Thekla Hamm, and Mirko H. Wagner;
licensed under Creative Commons License CC-BY 4.0

1 Introduction

Drawings of graphs are encountered early and used often: children are challenged to draw $K_{3,3}$ without crossings, drawings of graphs are regularly used when teaching graphs, they appear in any graph-related textbook, and we use them in our research discussions. Drawings of graphs naturally led to the concept of planar graphs and, more generally, an interest to control the crossings in the drawings.

Planar graphs enjoy a rich structural theory, have spurred research in graph theory and graph algorithms, and are relatively well understood. However, they are a quite restrictive class of graphs, and non-planar graphs still have to be drawn. For this reason, a range of beyond-planarity concepts have been suggested and investigated [12, 21, 29]. In this work, we will focus on 1-planarity, one of the extensions of planarity that has attracted much interest [22]. A drawing of a graph is 1-*planar* (or has *local crossing number* 1) if each edge participates in at most one crossing and adjacent edges do not cross[1]. A graph is 1-planar iff it admits a 1-planar drawing.

Generally speaking, the class of 1-planar graphs is not well understood. We know that recognizing 1-planar graphs is computationally hard [16, 23], even for graphs that are obtained from a planar graph by adding a single edge [9], or for graphs with constant treewidth, pathwidth or even bandwidth [1]. A fixed-parameter algorithm for deciding the existence of 1-planar drawings has been obtained recently by Hamm and Hlinený [19] (and actually is an easy consequence of [17]) using the total number of crossings in the drawing as a parameter.

Biedl and Murali [4] noticed that the crossings in a 1-planar drawing can be classified into different types, and the (non-)existence of some types may have important consequences. We will call these different types *crossing types*, and they describe adjacencies between the endpoints of the edges involved in that crossing. Consequently, there are six different crossing types (up to symmetry), for which we follow the terminology from [4]: (1) A *full* crossing is one in which each endpoint of an edge involved in that crossing is connected to each other such endpoint ($\boxtimes$), (2) an *almost full* crossing is one in which all but one pair of endpoints of the edges involved in that crossing are connected to each other ($\bowtie$), (3) a *bowtie* crossing is one in which the graph induced by the endpoints of the edges involved in that crossing is a 4-cycle ($\bowtie$), (4) an *arrow* crossing is one in which there is precisely one endpoint of an edge involved in that crossing that is connected to all other such endpoints ($\trianglerighteq$), (5) a *chair* crossing is one in which all but one pair of endpoints of the edges involved in that crossing are independent if one were to remove the edges involved in the crossing ($\barwedge$) (6) a $\times$ crossing is one in which all endpoints of the edges involved in that crossing are independent if one were to remove the edges involved in the crossing ($\times$).

For each non-empty $\mathcal{S} \subseteq \{\boxtimes, \bowtie, \bowtie, \trianglerighteq, \barwedge, \times\}$, we say that a drawing is $\mathcal{S}$-restricted 1-planar if it is 1-planar and the type of each crossing belongs to $\mathcal{S}$, and a graph is $\mathcal{S}$-restricted 1-planar if it admits a $\mathcal{S}$-restricted 1-planar drawing.

We already know that limiting the crossing types can have consequences. For instance, Biedl and Murali [4] described an algorithm to compute the vertex connectivity of a graph that is given with a 1-planar drawing without $\times$ crossings. This was extended by Biedl, Bose and Murali [3] to allow some $\times$ crossings in a controlled manner. Bose et al. [7] have shown that graphs that have a 1-planar $\times$-crossing-free drawing have bounded cop-number. However, it was not clear how difficult is to decide if a given graph has such a 1-planar drawing without $\times$ crossings, and this is posed explicitly as an open problem in [4].

[1] The condition of non-crossing adjacent edges is usually added because whenever the edges uv and uv' cross in a drawing, we can get a drawing with fewer crossings by redrawing the start of the edges uv, uv'.

On the other end of the spectrum when it comes to crossing types, Brandenburg [8] has shown that the class of 1-planar graphs that admit 1-planar drawings where *all* crossings are ⊠ crossings is equivalent to the class of 4-map graphs with holes, and thus {⊠}-restricted 1-planar graphs are recognizable in polynomial time. Münch and Rutter [26] show that, for each subset $\mathcal{S}$ of crossing types, recognizing $\mathcal{S}$-restricted 1-planar is FPT with respect to the *number* of crossings.

Another important thread in our work is that of straight-line drawings, which we refer to as *geometric drawings*. It is well-known that planar graphs admit a geometric planar drawing [15, 28]. However, we know that whether in the drawing we use arbitrary curves for the edges or we use straight-line segments does affect the crossings [5]. Similar phenomena occur for 1-planar graphs: not all graphs that have a 1-planar drawing have a geometric 1-planar drawing. Thomassen [27] provided a criterion to know when a 1-planar drawing can be continuously deformed into an equivalent 1-planar drawing with straight-line edges. The characterization is based on forbidden configurations in the drawing; we will explain this in detail below. How to transform efficiently a 1-planar drawing into a geometric one has been considered by Hong et al. [20].

Our contribution. We systematically investigate the complexity of recognizing $\mathcal{S}$-restricted 1-planar graphs for every fixed, nonempty subset $\mathcal{S}$ of crossing types. Fortunately, the characterization is neat, as it does not need to go over the 63 possible cases, and shows an interesting dichotomy when considering graphs parameterized by the treewidth:

- If $\mathcal{S} \subseteq \{\boxtimes, \bowtie, \bowtie\}$, it is FPT parameterized by the treewidth to recognize $\mathcal{S}$-restricted 1-planar graphs.
- If $\mathcal{S} \cap \{\diagup\!\!\diagdown, \times, \times\} \neq \emptyset$, then it is NP-hard to recognize 1-planar graphs that are $\mathcal{S}$-restricted 1-planar, even for graphs that have the treewidth bounded by constant.

Thereby we do not only resolve the open question from [4] (negatively), but we also obtain a very detailed perspective on how hard is recognizing 1-planar graphs with constant treewidth that admit 1-planar drawings with certain crossing types.

We also show that our dichotomy extends to the geometric setting, i.e. when all edges are required to be drawn as straight-line segments. Surprisingly, the same characterization works. However, extending the result to the geometric setting requires substantial effort.

Techniques. We use the fact that we target treewidth parameterizations to be able to invoke Courcelle's theorem [10]. However, expressing 1-planarity is something that is not possible efficiently in MSO (the logic required to formulate a problem in so that Courcelle's theorem implies a fixed-parameter algorithm), unless FPT = W[1]. This is where restricting crossing types becomes essential. We are able to show that restricting to crossing types that force 4-cycles to be present on the endpoints of the crossing edges allows for encoding 1-planarity in MSO in a constant-length formula. The key insight to do so is to realize an elegant way in which we can identify pairs of crossing edges without explicitly coding them into an MSO-formula.

To adapt this to the geometric setting, we use the fact that small topological obstructions to geometric 1-planarity are known. As MSO cannot speak about topology, it is useful that we can first reduce to instances whose prospective planarizations have unique embeddings.

Our NP-hardness results are all based on a single reduction from 3-Partition which are modularly adaptable for each crossing type that does not contain a 4-cycle.

2 Preliminaries

We use standard terminology and notation from graph theory [13] and assume general familiarity with parameterized complexity theory [11, 14]. We explain here some concepts and conventions that perhaps are encountered less often.

Graph terminology. All graphs we consider are simple and undirected. A separator S induces a *separation* $(X_1, \dots, X_\ell)$ for some $\ell \in \mathbb{N}$ if $\cup_{1 \le i \le \ell} X_i = V(G)$ and, for $1 \le i \ne j \le \ell$, we have $X_i \cap X_j = S$ and $S \subsetneq X_i \subset V(G)$. Given a separator S a separation is not necessarily unique.

We call G *internally 3-connected* if G can be obtained from a 3-connected graph G' by subdividing (with single vertices) a subset of edges of G'. (Here G' may have some parallel edges and at most one edge in each group of parallel edges is not subdivided.) The only 2-separators in an internally 3-connected graph are neighborhoods of degree-2 vertices.

If G is a connected graph then a *block-tree* or *BC-tree* of G is a tree T whose vertices are cutvertices and blocks of G, and adjacency in T is defined as containment in G.

A tree T is the *SPR-tree* of a 2-connected graph G, if every node t of T is labeled with S-, P-, or R-label, and it represents a graph, called the *skeleton of* t and denoted $\mathrm{sk}(t)$, whose edges are labeled *real* and *virtual*. S-nodes represent cycles, P-nodes represent dipoles, and R-nodes represent 3-connected graphs. Every edge $xy \in E(T)$ pairs a pair of directed virtual edges from $\mathrm{sk}(x)$ and $\mathrm{sk}(y)$, respectively, and every virtual edge e from $\mathrm{sk}(x)$ is paired with another virtual edge. We obtain G from its SPR-tree representation by identifying pairs of paired virtual edges and deleting the identified pairs. The graphs represented by R-nodes are called *3-connected components* of G. The *skeleton+* of an R-node x, denoted $\mathrm{sk}^+(x)$, is obtained from $\mathrm{sk}(x)$ by adding a real edge for every virtual edge that corresponds to an edge in G and subdividing once every virtual edge. Note that the skeleton+ of an R-node is internally 3-connected.

While BC-trees are a classical concept in graph theory, SP(Q)R-trees were first described by Di Battista and Tamassia [2]. Both BC-trees and SPR-trees of a graph G have size bounded by $O(|G|)$ and can be computed in linear time [18].

Graph drawing. A characterization of planar graphs, which will be useful to us, is that a graph is planar, if and only if it does not contain a subgraph that is a subdivision of K_5 or $K_{3,3}$ [24]; such subgraphs are also called *Kuratowski subdivisions*.

The *planarization* $\mathcal{G}^\times$ of a 1-planar drawing $\mathcal{G}$ of a graph G is obtained from $\mathcal{G}$ by replacing each pair of crossing edges with a new vertex positioned in a crossing which is linked to the endpoints of the crossing edges. Essentially this means we replace pairs of crossing edge segments by 4-claws.

A *face* of a drawing is a maximal connected subset of $\mathbb{R}^2$ that does not intersect the drawing (this implies that faces are always open sets, and this definition is also for drawings with crossings). For a simple closed curve in a drawing, a vertex is drawn *inside* that curve if it is separated from the outer face by that curve, and *outside* of that curve otherwise.

This allows us to give a useful equivalent characterization of geometric 1-planar drawings: Thomassen [27] showed that a drawing $\mathcal{G}$ can be transformed into a geometric 1-planar drawing by a homeomorphism of the plane if and only if it is 1-planar and neither of the following is contained in $\mathcal{G}$ (see Figure 1 for reference):

- Three edges ss', sb and $s'b'$ such that sb and $s'b$ cross and b and b' are inside the closed curve given by the drawings of ss', $s'b'$ from s' up to its crossing with sb and sb from its crossing with $s'b'$ up to s; such a situation is called a *B-configuration*.

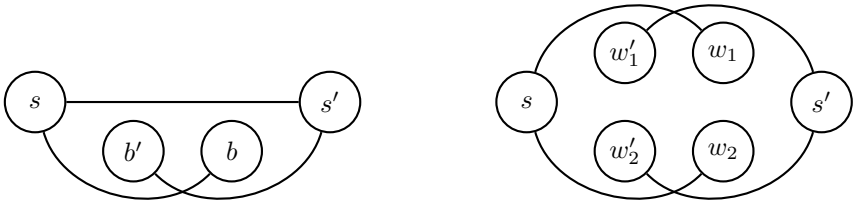


Figure 1 B- and W-configuration.

- Four edges sw_1, sw_2, $s'w'_1$ and $s'w'_2$ such that sw_1 and $s'w'_1$ cross and sw_2 and $s'w'_2$ cross and w_1, w_2, w'_1 and w'_2 lie inside the closed curve given by the drawings of the four edges between their crossings and s and s'; such a situation is called a *W-configuration*.

In both configurations above we call the pair of vertices s, s' the *spine*.

We overload the term '*boundary* of a face' to mean all the vertices, edges, edge segments and crossings whose drawings lie on the (pointwise set) boundary of a face. We say that two vertices are *cofacial* in a drawing if they lie together on the boundary of an arbitrary face of the drawing. It is known that in any 3-connected graph G a cycle $C \subseteq G$ is the boundary of a face in any planar drawing of G if and only if $G - V(C)$ is connected and C is an induced cycle of G; see for example [25, Theorem 2.5.1].

Parameterized complexity. Given a tree decomposition (T, χ) of a graph G, the graph $(V(G), \{uv \mid \exists t \in V(T)\ \{u, v\} \subseteq \chi(t)\}$ is called a *chordal completion* of G. By construction, G is a subgraph of any chordal completion, such a chordal completion has treewidth at most $\mathrm{tw}(T, \chi)$, and chordal completions are *chordal*, i.e. do not contain induced 4-cycles. Courcelle's theorem [10, 14] states that any problem encodable by an MSO_2-sentence φ over input graph G can be decided in FPT-time parameterized by $\mathrm{tw}(G) + |\varphi|$, where $|\varphi|$ is the length of the sentence φ.

3 Fixed-Parameter Tractable Cases by Treewidth

The aim of this section is to show the following theorems.

▶ **Theorem 1.** *It is* FPT *parameterized by treewidth to recognize $\mathcal{S}$-restricted* 1*-planar graphs if $\mathcal{S} \subseteq \{\boxtimes, \bowtie, \bowtie\}$.*

▶ **Theorem 2.** *It is* FPT *parameterized by treewidth to recognize geometric $\mathcal{S}$-restricted* 1*-planar graphs if $\mathcal{S} \subseteq \{\boxtimes, \bowtie, \bowtie\}$.*

Throughout this section, fix $\mathcal{S} \subseteq \{\boxtimes, \bowtie, \bowtie\}$. Observe that in this case every pair of edges defining a crossing lie on a common 4-cycle.

Parameterizing by treewidth in principle allows us to employ the powerful machinery of Courcelle's theorem which has previously been useful in the design of fixed-parameter algorithms for variants of the crossing number problem: Specifically in an MSO-formula one can quantify over a bounded number of pairs of subdivision vertices which should correspond to crossing vertices in the planarization of a desired drawing, ensuring that said planarization contains no Kuratowski subdivision (and in the geometric setting also no planarized B- or W-configuration). Explicitly quantifying over each pair of subdivision vertices which are "fused" into a crossing vertex results in such a formula's length being bounded in the number

of crossings, which is prohibitive for obtaining an FPT-algorithm via Courcelle's theorem if the number of crossings is not bounded in the parameter. It is obvious that this difficulty also applies to recognizing (geometric) $\mathcal{S}$-restricted 1-planar graphs.

On a high level, we overcome this difficulty by quantifying over an unpaired set of subdivision vertices (this can be done using a single variable) and deriving an unambiguous pairing of them using the 4-cycle on the endpoints of the corresponding edges that needs to exist because of the only crossing types we allow.

As a preparatory step for such an unambiguous pairing to be possible (how this is used will become clear in the proof of Lemma 9), we first argue that we can reduce to a setting in which we consider internally 3-connected graphs (at the expense of slightly generalizing the problem we consider). This will be carried out in Section 3.1.

Then deriving a pairing and formulating an appropriate MSO-encoding is carried out in Section 3.2 for Theorem 1 and Section 3.2.2 for Theorem 2.

3.1 Reducing to Internally 3-Connected Instances

Through a sequence of reductions that manipulate drawings of subgraphs, one can reduce deciding $\mathcal{S}$-restricted 1-planarity to internally 3-connected graphs. We state the result here and invite the reader to the full version for details.

▶ **Theorem 3.** *A graph is $\mathcal{S}$-restricted 1-planar if and only if the skeleton+ of each of its 3-connected components is $\mathcal{S}$-restricted 1-planar.*

We can carry out the arguments with the goal of also arriving at internally 3-connected graphs in the geometric setting. Notice that because the choice of the outer face matters for geometric $\mathcal{S}$-restricted 1-planar drawings, we need to consider a generalization of the problem of recognizing $\mathcal{S}$-restricted 1-planar graphs that takes this into account appropriately.

A graph $G = (V, E)$ is *O-geometric $\mathcal{S}$-restricted 1-planar* for some $O \subset V$ with $|O| \leq 1$, if there is a geometric $\mathcal{S}$-restricted 1-planar drawing of G where the vertex from O, if it exists, is on the outer face. To ultimately arrive at internally 3-connected instances we need a sequence of lemmas. We show one of them, without proof, to showcase the nuances of the procedure. See [8, Lemma 1] for a similar statement.

▶ **Lemma 4.** *Let $G = (V, E)$ be a 2-connected graph, $O \subset V$ with $|O| \leq 1$, s a cutvertex. Further let $(X_1, \ldots, X_\ell)$ be a separation induced by s and $G_i = G[X_i]$ for $i \in \{1, \ldots, \ell\}$, such that s is not a cutvertex of G_i for any $i \in \{1, \ldots, \ell\}$. Then G is O-geometric $\mathcal{S}$-restricted 1-planar if and only if $O \subset G_i$ for some $i \in \{1, \ldots, \ell\}$ and both G_i is O-geometric $\mathcal{S}$-restricted 1-planar and for all $j \in \{1, \ldots, \ell\} \setminus \{i\}$, G_j is $\{s\}$-geometric $\mathcal{S}$-restricted 1-planar.*

▶ **Theorem 5.** *Given an* FPT*-algorithm parameterized by treewidth that decides whether internally 3-connected graphs are O-geometric $\mathcal{S}$-restricted 1-planar, we can formulate an* FPT*-algorithm parameterized by treewidth to decide whether general graphs are geometric $\mathcal{S}$-restricted 1-planar.*

Proof sketch. It is obvious that deciding geometric $\mathcal{S}$-restricted 1-planarity is equivalent to deciding $\emptyset$-geometric $\mathcal{S}$-restricted 1-planarity.

Using Lemma 4, we can formulate a leaves-to-root dynamic program along the BC-tree of our input graph G which makes calls to a subroutine that decides whether 2-connected graphs are O-geometric $\mathcal{S}$-restricted 1-planar for some given O of cardinality at most 1. Such a subroutine can in turn be formulated as a leaves-to-root dynamic program along the

SPR-tree of a 2-connected graph. If no R-node is present in the considered subtree, then the corresponding graph is planar and any specified vertex can be placed on the outer face. As soon as an R-node is present, we can dynamically reduce to internally 3-connected instances, which are essentially the skeleton+ of the R-node, and solve them using the FPT-algorithm for that type of instances. Overall, during the dynamic program, there are a linear number of calls made to the solver for internally 3-connected graphs and only a polynomial-time overhead apart from that. ◀

3.2 An MSO-Encoding

We first focus on the non-geometric setting and describe how to adapt our formula towards solving the geometric setting in Section 3.2.2. So, until then fix G to be an instance of $\mathcal{S}$-RESTRICTED 1-PLANAR such that G is internally 3-connected and $\mathrm{tw}(G) \leq k$.

Our strategy to show fixed-parameter tractability is by giving a constant-length MSO-encoding of a graph G being $\mathcal{S}$-restricted 1-planar (or a graph G with a singleton-set O of outer face required vertices being geometric $\mathcal{S}$-restricted 1-planar) on an auxiliary graph $\tilde{G}$ whose treewidth is not significantly larger than that of G.

Our MSO-encoding will express that there exist some pairs of edges which occur in some subgraph of G isomorphic to a graph in $\mathcal{S}$ such that if we assume these pairs are all pairs of crossing edges, the resulting planarization contains no Kuratowski subdivisions, i.e. is planar, or in the geometric setting additionally contains no B- or W-configuration.

Expressing the non-existence of a constant-size obstruction such as a Kuratowski subdivision in a planarization after explicitly identifying pairs of crossing edges is not difficult and has previously been done e.g. in the fixed parameter algorithm for (non-local) CROSSING NUMBER [17]. To be explicit, before applying Courcelle's theorem, one can subdivide each edge with a vertex corresponding to a possible crossing vertex in the planarization in case that edge is crossed. Then, given explicit access to pairs of crossing edges, the MSO-formula precludes the existence of the constant-size obstruction in the planarization by explicitly forbidding vertices and edges that can form that obstruction when the original edge relation of the graph is replaced by two vertices being endpoints of an original edge or an edge subdivision vertex for an edge e and a neighbor of the subdivision vertex for the edge e' for which e and e' are a pair of crossing edges.

However, we do not have explicit access to all pairs of crossing edges. In a setting where the total number of crossings is a parameter, this difficulty is easily overcome by quantifying with variables over possible crossing edge pairs. However, in this way the length of the formula depends on the number of such pairs and hence this approach is infeasible when there can be one crossing per edge. We will have to implicitly encode the pairs of crossing edges instead. For this, the restricted types of allowed crossings are critical.

3.2.1 Identifying planarized crossings

To build $\tilde{G}$, we augment G in two ways: Similarly as in [17], for every edge of G we insert a P_3 between its endpoints and label the new middle vertex with `cr`. In MSO, selecting a set of `cr`-vertices encodes that the corresponding edges are crossing edges.

However, merely selecting a set of crossing edges is insufficient to describe and then check planarity of a prospective planarization. We also need to identify which pairs of crossing edges cross each other. To pair up the crossing edges, ideally we would want MSO to be able to select for a crossing edge the two end vertices of the edge it crosses, e.g., by $\tilde{G}$ having a vertex between the four end vertices of every pair of edges that could possibly cross. But

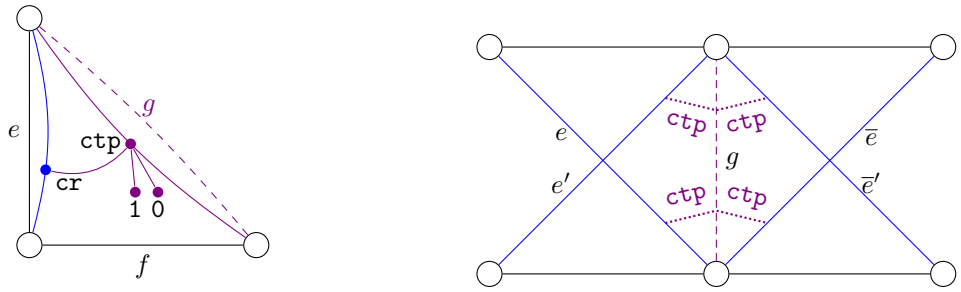


Figure 2 (left) chord-tied pair (e, g) with central vertex u and flag pendants, **(right)** chord edge with two crossings.

this is not possible to do in general without $\mathrm{tw}(\tilde{G})$ becoming unbounded in $\mathrm{tw}(G)$. Instead, we do the next best thing: We augment $\tilde{G}$ in such a way that MSO can select for every crossing edge one of the end vertices of the edge it crosses. Because in every crossing type $s \in \{\boxtimes, \bowtie, \bowtie\}$ there is a 4-cycle that includes both crossing edges of s, this essentially asks MSO to select for each crossing, two paths of length two that together form a 4-cycle.

To do so, we compute a tree decomposition (T, χ) of G of width $\mathcal{O}(\mathrm{tw}(G))$ [6] and use this to construct a chordal completion $\hat{G}$ of treewidth $\mathcal{O}(\mathrm{tw}(G))$. For every 4-tuple of edges $(e, f, e', f') \in E(G)^4$ forming a 4-cycle in G (in that order), the chordality of $\hat{G}$ implies that there is at least one chord edge for (e, f, e', f'): an edge $g \in E(\hat{G}) \setminus \{e, f, e', f'\}$ connecting two vertices of $e \cup f \cup e' \cup f'$. For each 4-cycle (e, f, e', f') whose vertices induce a graph underlying a crossing type in $\mathcal{S}$ where e and e' cross and each chord g for (e, f, e', f'), we iteratively add the following graph elements to $\tilde{G}$: A P_3 between the endpoints of g with a *new* middle vertex labeled `ctp` and an edge between the `ctp` vertex and the `cr`-vertex that was inserted for e. We call (e, g) a *chord-tied pair*, if e is not incident to a degree 2 vertex in G. We call e the *crossing edge* and g the *chord edge* of this chord-tied pair. Each chord-tied pair is uniquely identified by a `ctp`-vertex.

At this point, for a given chord-tied pair, there might be multiple other chord-tied pairs with the same chord edge. See, e.g., the right of Figure 2, where there are four chord-tied pairs (e, g), (e', g), $(\bar{e}, g)$, and $(\bar{e}', g)$ with the same chord edge g. This is why we add two *flag pendants* labeled `0` and `1` to each `ctp` vertex in $\tilde{G}$ to encode a boolean flag for every chord-tied pair; see the left side of Figure 2 for an illustration. A pair of edges $\{e, e'\}$ is called a *crossing pair* if e and e' are the crossing edges of two different chord-tied pairs with the same chord edge and the same flag. For this definition to be able to uniquely identify which edges cross, at most four chord-tied pairs can share a chord edge, but as we show in Lemma 9 this suffices for internally 3-connected $\mathcal{S}$-restricted 1-planar graphs.

This completes our construction of $\tilde{G}$. First we bound its treewidth.

▶ **Lemma 6.** $\mathrm{tw}(\tilde{G}) \in \mathcal{O}(\mathrm{tw}(G)^2)$.

The next definition allows us to associate chord tied pairs and flags with pairings of crossings.

▶ **Definition 7** (chord description). *A set* CD *of chord-tied pairs with flags (i.e., a set of tuples (e, g, b), where (e, g) is a chord-tied pair and $b \in \{0, 1\}$) is a* chord description *of G, if it satisfies the following conditions:*

1. *for every chord-tied pair in* CD *there is exactly one other chord-tied pair in* CD *with the same chord edge and flag; if (e, g, b) and (e', g, b) belong to* CD*, we say that $\{e, e'\}$ form a* crossing pair of CD*;*

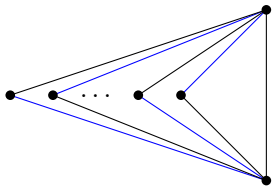


Figure 3 In a chordal graph arising from adding an even number of distinct triangles tips to a shared edge, the shared edge is the only chord and selecting the blue edges, i.e. the edges between the tip and one shared endpoint for one half of the triangles and the edges between the tip and the other shared endpoint for the other half allows for a quadratic number of possible crossing pairs, namely any pairing of independent blue edges. This ambiguity arises because this graph is only 2- and not 3-connected.

2. *every edge is the crossing edge of at most one chord-tied pair in* CD*;*
3. *the graph* G_{CD}*, where, starting with* G*, every crossing pair* $\{e, e'\}$ *of* CD *is replaced by a 4-claw on* $V(\{e, e'\})$*, is planar; and*
4. *every crossing pair* $\{e, e'\}$ *of* CD *can cross* $\mathcal{S}$*-restricted* 1*-planarly, i.e., the vertices in* $e \cup e'$ *induce a crossing type from* $\mathcal{S}$ *where* e *and* e' *cross.*

We can derive a drawing from a chord description as follows.

▶ **Lemma 8.** *Let* CD *be a chord description of* G*. Then there is an* $\mathcal{S}$*-restricted* 1*-planar drawing of* G *where only crossings pairs of* CD *cross.*

Proof. We start with a planar drawing $\mathcal{G}_{\mathrm{CD}}$ of G_{CD}, which exists by Cond. 3. Then, we un-replace in $\mathcal{G}_{\mathrm{CD}}$ every 4-claw corresponding to a crossing pair of CD by its original edges. This essentially corresponds to undoing a planarization and yields a drawing $\mathcal{G}$ of G where only crossing pairs of CD cross. (Notice that the un-replacement of 4-claws corresponding to a crossing pair of a $\bowtie$-crossing does not necessarily yield a crossing in $\mathcal{G}$ because the endpoints of the edges of the crossing pair might not alternate in the rotation around the center of the 4-claw, but for $\boxtimes$- and $\boxtimes$-crossings it always does.) By Cond. 2 and the definition of crossing pairs, each edge is crossed at most once in $\mathcal{G}$. By Cond. 4, each crossing in $\mathcal{G}$ is an $\mathcal{S}$-restricted 1-planar one. Therefore, $\mathcal{G}$ is an $\mathcal{S}$-restricted 1-planar drawing of G where only crossing pairs of CD cross. ◀

Conversely, we can derive a chord description from a hypothetical $\mathcal{S}$-restricted 1-planar drawing of G. Here the internal 3-connectivity of G is important; see Figure 3.

▶ **Lemma 9.** *Let* G *be an internally* 3*-connected graph that has an* $\mathcal{S}$*-restricted* 1*-planar drawing. Then there exists a chord description* CD *such that for every chord-tied pair in* CD *there is an edge in* $\hat{G}$ *between its chord vertices.*

Proof. We do this proof by constructing CD from some $\mathcal{S}$-restricted 1-planar drawing $\mathcal{G}$ of G. One can show that, if G is an internally 3-connected graph that has an $\mathcal{S}$-restricted 1-planar drawing, then there is an $\mathcal{S}$-restricted 1-planar drawing of G where no edge participating in a crossing has an endpoint of degree 2. We consider such a drawing $\mathcal{G}$. For every crossing in $\mathcal{G}$ we can construct two (different) chord-tied pairs with the same chord edge and the same flag, because the pair of edges is on a 4-cycle which has to have a chord in $\hat{G}$. By Cond. 1 of being a chord description, the construction directly guarantees that each chord-tied pair has at least one partner. If we can show that every chord-tied pair has exactly one partner, then CD trivially also satisfies Cond. 2–4. In $\mathcal{G}$ two crossings whose corresponding chord-tied

pairs have the same chord vertices enclose a region that only consists of edges that already cross and this face is only incident to the chord vertices. Having a third such crossing would force one of the three crossings to be completely inside such a region. By G being internally 3-connected this can only happen if the crossing edges have degree-2 endpoints, which is not the case in $\mathcal{G}$. Therefore, we can choose the flags such that there are at most two chord-tied pairs in CD that share both the flag and the chord edge. ◀

Building an MSO_2-formula. The intuition behind the forthcoming formula is to select a subset D of `ctp`-vertices, which is equivalent to selecting a subset of chord-tied pairs, and select for each `ctp`-vertex a unique flag `0` or `1` such that the resulting chord-tied pairs with flags form a chord description. Thus, the general structure of the MSO_2-formula φ which we want to apply Courcelle's theorem to will be as follows:

$$\varphi := \exists D \exists M \; \forall d \in D \; \texttt{ctp}(d) \wedge \forall m \in M \; \texttt{0}(m) \vee \texttt{1}(m) \tag{1}$$

$$\wedge \forall d \in D \; \big(\exists m \in M \; \textsc{adj}(d,m) \wedge \forall m' \in M \; (\textsc{adj}(d,m') \Rightarrow m = m')\big) \tag{2}$$

$$\wedge \forall m \in M \; \exists d \in D \; \textsc{adj}(d,m) \tag{3}$$

$$\wedge \textsc{ChordDescription}(D, M), \tag{4}$$

where $\textsc{ChordDescription}(D, M)$ encodes that D with each chord tied pair corresponding to $d \in D$ receiving the flag coinciding with the label of the unique (by (2)) neighbor of d in M is a chord description of G.

The encoding of $\textsc{ChordDescription}(D, M)$ is done formulating each of the conditions from Definition 7. The description is technical and tedious; we refer the reader to the full version.

▶ **Lemma 10.** *Definition 7 holding for a set D of chord-tied pairs if flagged according to a unique matching to a set M of flag pendants can be encoded as a constant-length MSO_2-formula with free variables D and M.*

Combining Lemmata 8–10 we easily obtain the following.

▶ **Corollary 11.** *Let G be an internally 3-connected graph. Then $\tilde{G} \models \varphi$ if and only if G is a yes-instance for $\mathcal{S}$-*RESTRICTED 1-PLANAR.

Since the formula φ in Corollary 11 has constant length and $\mathrm{tw}(\tilde{G}) \in \mathcal{O}(\mathrm{tw}(G)^2)$ (because of Lemma 6), we can apply Courcelle's theorem to conclude that for internally 3-connected graphs there is an FPT-algorithm parameterized by treewidth to recognize whether they are $\mathcal{S}$-restricted 1-planar graphs. Theorem 1 now follows from Theorem 3.

3.2.2 Encoding Straightline Planarity

Now we turn to the geometric setting, i.e. recognizing $\mathcal{S}$-restricted 1-planar graphs. One difference is that we need to additionally express the non-existence of planarized B- and W-configurations in the graph corresponding to the planarization of a $\mathcal{S}$-restricted 1-planar drawing. Further, by Lemma 4 and other similar-looking statements to reduce the problem to the case of internally 3-connected graphs, we need to additionally allow requiring the containment of some distinguished vertex on the outer face.

For expressing outer-face requirement, we consider the distinguished vertex colored with `outer` in the graph when we apply Courcelle's theorem if there is any.

For expressing the non-existence of planarized B- and W-configurations, notice that quantifying over all abstract graphs underlying them is trivially possible in MSO.

To forbid the actual drawings that constitute B- and W-configurations, it will be convenient to consider a certain kind of well-behaved geometric $\mathcal{S}$-restricted 1-planar drawings.

▶ **Definition 12.** *A geometric $\mathcal{S}$-restricted 1-planar drawing $\mathcal{G}$ is* crossing confined *if for each crossing, say between edges e and e', no other edge between any pair of vertices among the endpoints of e and e' is involved in a crossing.*

▶ **Observation 13.** *If there is a geometric $\mathcal{S}$-restricted 1-planar drawing of G then there is also a crossing confined one. Every outer face vertex in the old drawing is also an outer face vertex in the new drawing.*

Proof. Assume v_1v_2, u_1u_2 is an arbitrary pair of mutually crossing edges in $\mathcal{G}$. We want to find a geometric $\mathcal{S}$-restricted 1-planar drawing in which none of the (potentially existing) edges $v_1u_1, u_1v_2, v_2u_2, u_2v_1$ are crossed. Without loss of generality we may focus on vertices u_1 and v_1 and assume that $u_1v_1 \in E(G)$. Let c be the vertex in the planarization $\mathcal{G}^\times$ corresponding to the crossing of edges v_1v_2 and u_1u_2. Note that vertices v_1 and u_1 are consecutive around c in the planarization $\mathcal{G}^\times$ and are therefore cofacial in $\mathcal{G}$. The edge u_1v_1, if crossed, can be redrawn without a crossing. Doing this at an epsilon distance from *some* pairwise crossing edges incident to u_1 and v_1, respectively (which is not necessarily the pair v_1v_2, u_1u_2), does not introduce new B- or W-configurations, so the resulting drawing can be geometrically realized. ◀

If we assume crossing confinedness of a targeted solution, which we can do without loss of generality, then the only way in which the abstract graphs underlying planarized B- and W-configurations can occur in a geometric 1-planar drawing is if a specific face is not the outer face. We provide a lemma for each configuration.

▶ **Lemma 14.** *A crossing confined 1-planar drawing $\mathcal{H}$ of an internally 3-connected graph H contains a B-configuration on edges ss', sb and $s'b'$ in which sb and $s'b'$ cross if and only if the crossing between sb and $s'b'$ together with s and s' are precisely the set of vertices on the boundary of the outer face of $\mathcal{H}^\times - \{v \in V(G) \setminus \{b, b'\} \mid N(v) = \{s, s'\}\}$.*

Proof. $\Rightarrow$ Because $\mathcal{H}$ is 1-planar there are no edges crossing sb other than $s'b'$ and vice versa. Because $\mathcal{H}$ is crossing confined, ss' is not involved in any crossing. This means that if there are vertices drawn outside of the closed curve given by ss' and $s'b'$ from s' to its crossing and sb from its crossing to s, then they must be connected to b only by paths through s or s'. However, this contradicts the fact that H is internally 3-connected unless the vertices are of degree 2 and only neighboring s and s', implying that there are no other vertices outside of that curve and it has to be the boundary of the outer face.

$\Leftarrow$ By contraposition, if s, s', and the crossing between sb and $s'b'$ are the only vertices on the outer face of $\mathcal{H}^\times$ after deleting all degree-2 vertices with neighbors exactly s, s' then b and b' are drawn inside the curve given by ss' and $s'b'$ from s' to its crossing and sb from its crossing to s. This implies a B-configuration in $\mathcal{H}$. ◀

Without even using crossing confinedness (but just the fact that in 1-planar drawings each edge is involved in at most one crossing), one can show a similar statement for W-configurations.

▶ **Lemma 15.** *A crossing confined 1-planar drawing $\mathcal{H}$ of an internally 3-connected graph H contains a W-configuration with spine s, s' on edges sw_1, sw_2, $s'w_1'$ and $s'w_2'$ in which sw_1 and sw_2 and also $s'w_1'$ and $s'w_2'$ cross each other if and only if their crossings together with s and s' are precisely the set of vertices on the boundary of the outer face of $\mathcal{H}^\times - \{v \in V(G) \setminus \{w_1, w_2, w_1', w_2'\} \mid N(v) = \{s, s'\}\}$.*

Overall, this means we can expand φ from the previous subsection as follows for the geometric setting:

$$\begin{aligned}\varphi_{\text{geom}} := \varphi \wedge \text{CR-CONFINED}(D,F) \wedge \\ \big(\exists C\ \text{FACECYCLE}(C) \\ \wedge (\forall v\ \texttt{outer}(v) \Rightarrow (v \in C \vee (\exists n_1, n_2 \in C \forall w\ \text{ADJ}(v,w) \Rightarrow w = n_1 \vee w = n_2)) \\ \wedge (\forall b_1, b_2, b_3, b_4, b_5, c\ \text{B-CONFIG}(b_1, \dots, b_5, c) \Rightarrow C \neq \{b_1, b_2, c\}) \\ \wedge (\forall b_1, b_2, b_3, b_4, b_5, b_6, c_1, c_2\ \text{W-CONFIG}(b_1, \dots, b_6, c_1, c_2) \Rightarrow C \neq \{b_1, b_2, c_1, c_2\})\big),\end{aligned}$$

where if from the flagged chord-description quantified over in φ we get CD,

- CR-CONFINED(D,F) expresses that for each crossing pair $\{e, e'\}$ of the chord description CD encoded by (D,F), there is no crossing pair of CD containing another edge between any pair of endpoints of e and e'.
- FACECYCLE(C) expresses that $G_{\text{CD}}[C]$ is an induced non-separating cycle after the contraction of degree-2 vertices.
- B-CONFIG$(b_1, \dots, b_4, c)$ expresses the existence of the underlying graph of a planarized and uncrossed-edge-subdivided B-configuration with crossing vertex c and uncrossed subdivided edge b_1b_2 in G_{CD}.
- W-CONFIG$(b_1, \dots, b_6, c_1, c_2)$ expresses the existence of the underlying graph of a planarized W-configuration with crossing vertices c_1 and c_2 and shared endpoints of original edges b_1 and b_2 in G_{CD}.

It is easy to see that the above can be expressed as a constant-length MSO-formula (where the main difficulty lies in encoding the set of crossing pairs of the chord description encoded by (D,F) and G_{CD} which we argued how to do in the previous subsection.

▶ **Lemma 16.** *Let G be an internally 3-connected graph. Then $\tilde{G} \models \varphi_{geom}$ if and only if G is geometric $\mathcal{S}$-restricted 1-planar and there is a geometric $\mathcal{S}$-restricted 1-planar drawing of G in which all* `outer`*-colored vertices are on the outer face.*

Since the formula φ in Lemma 16 has constant length and $\text{tw}(\tilde{G}) \in \mathcal{O}(\text{tw}(G)^2)$ (because of Lemma 6), we can apply Courcelle's theorem to conclude that there is an FPT-algorithm parameterized by treewidth that decides whether internally 3-connected graphs are O-geometric $\mathcal{S}$-restricted 1-planar. Theorem 2 now follows from Theorem 5.

4 Hard Cases for Constant Pathwidth

In this section we show that for the remaining subsets of crossing types, the recognition problem becomes NP-complete in the geometric and non-geometric case, even if the graph has constant pathwidth.

▶ **Theorem 17.** *For $\mathcal{S} \cap \{\bowtie, \times, \times\} \neq \emptyset$ deciding geometric $\mathcal{S}$-restricted 1-planarity and $\mathcal{S}$-restricted 1-planarity is* NP*-complete and para-*NP*-hard when parameterized by the pathwidth.*

NP membership follows due to the fact that geometric 1-planar graphs are characterized by having drawings without B- and W-configurations. For NP-hardness, we show for every singleton $\mathcal{S}' \subset \{\bowtie, \times, \times\}$ how to transform any 3-PARTITION instance I into a graph with constant pathwidth that is geometric $\mathcal{S}'$-restricted 1-planar if I is a yes-instance and not even 1-planar if I is a no-instance. This then directly yields a reduction from 3-PARTITION to recognizing $\mathcal{S}$-restricted 1-planarity and geometric $\mathcal{S}$-restricted 1-planarity for any $\mathcal{S}$ with $\mathcal{S} \cap \{\bowtie, \times, \times\} \neq \emptyset$.

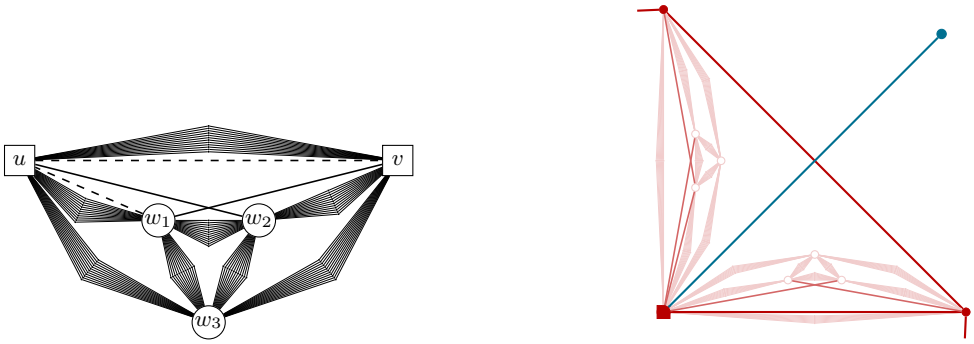


Figure 4 (left) Fence $F_{u,v}$ between nodes u and v. The single edges are thick; the direct edges whose presence depends on $\mathcal{S}'$ are dashed. **(right)** Example of how a splitter edge (blue) crosses a rim edge. All three crossings are ⧖-crossings.

Our transformation builds upon Grigoriev and Bodlaender's proof that determining 1-planarity is NP-complete [16]. We provide the reduction in detail, as it showcases the main ideas, and refer to the full version for complete proofs.

The problem 3-Partition is given a set A of $3m$ elements with $m \geq 3$, a bound $B \in \mathbb{Z}^+$, and a size $s(a) \in \mathbb{Z}^+$ for each $a \in A$ such that $\sum_{a \in A} s(a) = mB$, and asks whether A can be partitioned into m disjoint subsets $A_1, \ldots, A_m$ of size three such that for $1 \leq i \leq m$ we have $\sum_{a \in A_i} s(a) = B$. As 3-Partition is strongly NP-complete, we can assume that B is polynomial in m. For convenience, we furthermore assume for $\mathcal{S}' = \{⧖\}$ that both B and m are even.

Before we describe the transformation itself, we construct what we call *fences*. For any two nodes u and v, the fence $F_{u,v}$ between them is constructed as follows (see the left side of Figure 4 for reference): We start with a K_5 on u, v and three new nodes w_1, w_2, w_3. Next we replace all of its edges other than uw_2 and vw_1 by a *bundle* of $\ell = 12$ parallel paths of length two. We call uw_2 and vw_1 the *single edges* of the fence. For $\mathcal{S}' = \{⧅\}$ we re-add the edges uv and uw_1 and for $\mathcal{S}' = \{⧖\}$ we re-add either the edge uv or uw_1 (details follow later). This ensures that the crossing between the single edges is an $\mathcal{S}'$-crossing. The drawings depicted in the left side of Figure 4 shows a geometric $\mathcal{S}$-restricted 1-planar drawing of a fence where u and v share a face. The following lemma shows that we can treat fences as uncrossable edges in the context of 1-planar drawings.

▶ **Lemma 18.** *Let $G = (V, E)$ be a 1-planar graph and $u, v \in V$ be two nodes between which there is a fence $F_{u,v} \subseteq G$. Then in every 1-planar drawing Γ of G, the single edges of $F_{u,v}$ cross (and whereby u and v share a face).*

Given a 3-Partition instance I we construct a graph G_I as follows (see also Figure 5): We start with two wheels with fences as radians; one of size $3m$, the *transmitter*, and one of size Bm, the collector. For $\mathcal{S}' = \{⧖\}$ every even-numbered radian fence has the edge uv and every odd-numbered has uw_1. Next, we add a path of three fences between the $3i$-th node of the transmitter and the Bi-th node of the collector for $1 \leq i \leq m$. We call the resulting m paths of fences between the wheel centers *dividers*. For each element $a \in A$ we add a *splitter* Q_a, which consists of a claw of size $s(a)$ of fences and an edge between every degree-1 node of the claw and the collector center and an edge between the degree-$s(a)$ node of the claw and the transmitter center. One can show that I can be satisfyingly partitioned if and only if G_I is geometric $\mathcal{S}'$-restricted 1-planar. Moreover, the graph G_I has constant pathwidth. This leads to Theorem 17.

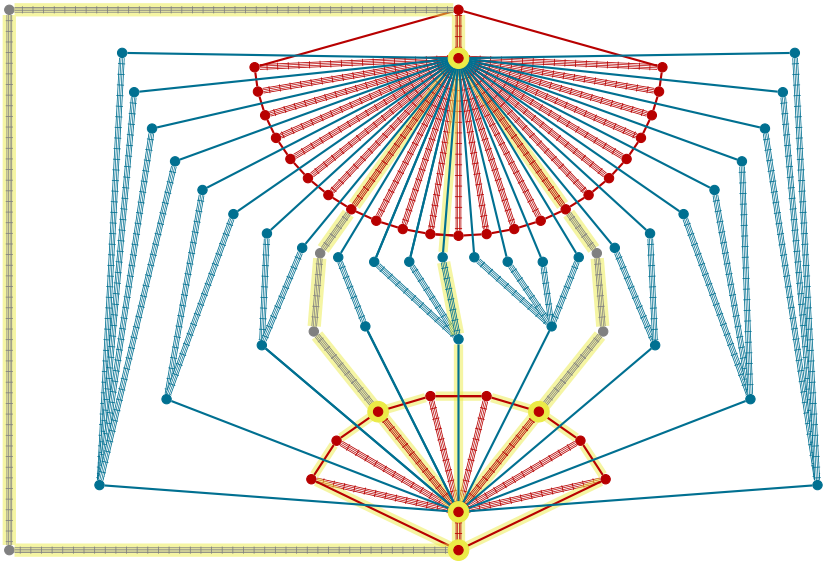


Figure 5 Graph G_I based on 3-Partition Instance I with $A = \{1, 2, 2, 2, 3, 3, 3, 4, 4\}$, $B = 10$, and $m = 3$. The fences have a fence pattern, the wheels are red, the splitters are blue, and the dividers are gray. A Kuratowski subdivision of interest is highlighted in yellow.

References

1 Michael J. Bannister, Sergio Cabello, and David Eppstein. Parameterized complexity of 1-planarity. *J. Graph Algorithms Appl.*, 22(1):23–49, 2018. doi:10.7155/JGAA.00457.

2 Giuseppe Di Battista and Roberto Tamassia. Incremental planarity testing (extended abstract). In *30th Annual Symposium on Foundations of Computer Science*, pages 436–441. IEEE Computer Society, 1989. doi:10.1109/SFCS.1989.63515.

3 Therese Biedl, Prosenjit Bose, and Karthik Murali. A parameterized algorithm for vertex and edge connectivity of embedded graphs. In *32nd Annual European Symposium on Algorithms, ESA 2024*, volume 308 of *LIPIcs*, pages 24:1–24:15, 2024. doi:10.4230/LIPICS.ESA.2024.24.

4 Therese Biedl and Karthik Murali. On computing the vertex connectivity of 1-plane graphs. In *50th International Colloquium on Automata, Languages, and Programming (ICALP 2023)*, volume 261 of *LIPIcs*, pages 23:1–23:16, 2023. doi:10.4230/LIPIcs.ICALP.2023.23.

5 Daniel Bienstock and Nathaniel Dean. Bounds for rectilinear crossing numbers. *J. Graph Theory*, 17(3):333–348, 1993. doi:10.1002/JGT.3190170308.

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 Prosenjit Bose, Jean-Lou De Carufel, Anil Maheshwari, and Karthik Murali. On 1-planar graphs with bounded cop-number. *Theor. Comput. Sci.*, 1037:115160, 2025. doi:10.1016/J.TCS.2025.115160.

8 Franz J. Brandenburg. On 4-map graphs and 1-planar graphs and their recognition problem. *CoRR*, abs/1509.03447, 2015. arXiv:1509.03447.

9 Sergio Cabello and Bojan Mohar. Adding one edge to planar graphs makes crossing number and 1-planarity hard. *SIAM J. Comput.*, 42(5):1803–1829, 2013. doi:10.1137/120872310.

10 Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Inform. and Comput.*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

11 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.

12 Walter Didimo, Giuseppe Liotta, and Fabrizio Montecchiani. A survey on graph drawing beyond planarity. *ACM Comput. Surv.*, 52(1):4:1–4:37, 2019. doi:10.1145/3301281.

13 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

14 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

15 István Fáry. On straight-line representation of planar graphs. *Acta Sci. Math. (Szeged)*, 11:229–233, 1948.

16 Alexander Grigoriev and Hans L. Bodlaender. Algorithms for graphs embeddable with few crossings per edge. *Algorithmica*, 49(1):1–11, 2007. doi:10.1007/S00453-007-0010-X.

17 Martin Grohe. Computing crossing numbers in quadratic time. *J. Comput. Syst. Sci.*, 68(2):285–302, 2004. doi:10.1016/J.JCSS.2003.07.008.

18 Carsten Gutwenger and Petra Mutzel. A linear time implementation of SPQR-trees. In *Graph Drawing, 8th International Symposium, GD 2000*, volume 1984 of *Lecture Notes in Computer Science*, pages 77–90, 2000. doi:10.1007/3-540-44541-2_8.

19 Thekla Hamm and Petr Hlinený. Parameterised partially-predrawn crossing number. In *38th International Symposium on Computational Geometry, SoCG 2022*, volume 224 of *LIPIcs*, pages 46:1–46:15, 2022. doi:10.4230/LIPICS.SOCG.2022.46.

20 Seok-Hee Hong, Peter Eades, Giuseppe Liotta, and Sheung-Hung Poon. Fáry's theorem for 1-planar graphs. In *Computing and Combinatorics - 18th Annual International Conference, COCOON 2012*, volume 7434 of *Lecture Notes in Computer Science*, pages 335–346, 2012. doi:10.1007/978-3-642-32241-9_29.

21 Seok-Hee Hong and Takeshi Tokuyama, editors. *Beyond Planar Graphs, Communications of NII Shonan Meetings*. Springer, 2020. doi:10.1007/978-981-15-6533-5.

22 Stephen G. Kobourov, Giuseppe Liotta, and Fabrizio Montecchiani. An annotated bibliography on 1-planarity. *Comput. Sci. Rev.*, 25:49–67, 2017. doi:10.1016/J.COSREV.2017.06.002.

23 Vladimir P. Korzhik and Bojan Mohar. Minimal obstructions for 1-immersions and hardness of 1-planarity testing. *J. Graph Theory*, 72(1):30–71, 2013. doi:10.1002/JGT.21630.

24 Kazimierz Kuratowski. Sur le probleme des courbes gauches en topologie. *Fundamenta mathematicae*, 15(1):271–283, 1930.

25 Bojan Mohar and Carsten Thomassen. *Graphs on surfaces*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, Baltimore, MD, 2001.

26 Miriam Münch and Ignaz Rutter. Parameterized algorithms for beyond-planar crossing numbers. In *32nd International Symposium on Graph Drawing and Network Visualization, GD 2024*, volume 320 of *LIPIcs*, pages 25:1–25:16, 2024. doi:10.4230/LIPICS.GD.2024.25.

27 Carsten Thomassen. Rectilinear drawings of graphs. *J. Graph Theory*, 12(3):335–341, 1988. doi:10.1002/JGT.3190120306.

28 Klaus Wagner. Bemerkungen zum Vierfarbenproblem. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 46:26–32, 1936.

29 Meirav Zehavi. Parameterized analysis and crossing minimization problems. *Comput. Sci. Rev.*, 45:100490, 2022. doi:10.1016/J.COSREV.2022.100490.

Realization of Temporally Connected Graphs Based on Degree Sequences

Arnaud Casteigts
Department of Computer Science, University of Geneva, Switzerland
LaBRI, Université de Bordeaux, France

Michelle Döring
Hasso Plattner Institute, University of Potsdam, Germany

Nils Morawietz
Institute of Computer Science, Friedrich Schiller University Jena, Germany
LaBRI, Université de Bordeaux, France

Abstract

Given an undirected graph G, the problem of deciding whether G admits a simple and proper time-labeling that makes it temporally connected is known to be NP-hard (Göbel *et al.*, 1991). In this article, we relax this problem and ask whether a given degree sequence can be realized as a temporally connected graph. Our main results are a complete characterization of the feasible cases, and a recognition algorithm that runs in $\mathcal{O}(n)$ time for graphical degree sequences (realized as simple temporal graphs) and in $\mathcal{O}(n+m)$ time for multigraphical degree sequences (realized as non-simple temporal graphs, where the number of time labels on an edge corresponds to the multiplicity of the edge in the multigraph). In fact, these algorithms can be made constructive at essentially no cost. Namely, we give a constructive $\mathcal{O}(n+m)$ time algorithm that outputs, for a given (multi)graphical degree sequence $\mathbf{d}$, a temporally connected graph whose underlying (multi)graph is a realization of $\mathbf{d}$, if one exists.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Mathematics of computing → Discrete mathematics; Networks → Network design and planning algorithms

Keywords and phrases temporal paths, gossiping, (multi)graphical degree sequences, edge-disjoint spanning trees, linear time algorithms

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.17

Related Version *Full Version*: `https://arxiv.org/abs/2504.17743`

Funding *Arnaud Casteigts*: Supported by the Swiss NSF, project RECAPT (200021-236640).
Michelle Döring: German Federal Ministry for Education and Research (BMBF) through the project "KI Servicezentrum Berlin Brandenburg" (01IS22092).
Nils Morawietz: Supported by the French ANR, project ANR-22-CE48-0001 (TEMPOGRAL).

Proof of statements marked with (⋆) are deferred to the full version.

1 Introduction

The problem of assigning time labels to the edges of a graph in such a way that every vertex can reach every other vertex by a non-decreasing path (temporal path) is known as the gossiping problem. Problems of this type were extensively studied in the late 70s - early 90s (see, e.g. [21, 22]), initially motivated by optimal scheduling of phone calls, where the agents are represented by vertices and the calls by time-labeled edges (e.g. [17, 6]). Most of the research in this line consider interactions that are mutually exclusive (an agent cannot have multiple phone calls at the same time) and non-repetitive (two agents cannot call each other twice). One of the landmark results in this area was obtained in 1991 by Göbel et al. [16], who showed that deciding whether a given graph admits such a labeling is NP-complete.

© Arnaud Casteigts, Michelle Döring, and Nils Morawietz;
licensed under Creative Commons License CC-BY 4.0

The past decade have seen a renewed interest in this type of problems, motivated by the modeling of various types of dynamic networks, such as wireless networks, social networks, autonomous vehicles, and transportation scheduling. In today's terminology, the gossip problem can naturally be rephrased in terms of temporal graphs. Formally, a temporal graph is a pair $\mathcal{G} = (G, \lambda)$ where $G = (V, E)$ is the underlying graph (in this work, undirected) and λ is a mapping that assigns time labels to the edges. Such a labeling is *simple* if every edge of G has a single time label, and it is *proper* if adjacent edges do not share any time label. The graph $\mathcal{G}$ is *temporally connected* (i.e. in class TC) if there exists a path traversing edges with non-decreasing times (temporal path) between each ordered pair of nodes. In this terminology, the gossip problem corresponds to deciding, given a graph G, if there exists a simple and proper labeling λ such that $\mathcal{G} = (G, \lambda) \in \mathsf{TC}$ (and finding such a labeling).

A number of similar realizability questions have been considered recently in the temporal graph literature, focusing on labelings that satisfy additional prescribed properties. For example, several works investigate whether the labeling can ensure that the fastest temporal paths between vertices match (or is upper-bounded by) a given matrix of pairwise durations [24, 12, 27, 26, 28]. Another problem asks that the temporal paths realize a target reachability relation [11], under various restrictions of the labeling function (see [7, 8] for discussions on the resulting expressivity). Further studies consider minimizing the number of labels [1], minimizing reachability times [10], maximizing overall reachability sets [5], or avoiding temporal cycles [2], to name a few.

Broadly speaking, all these problems, and their non-temporal versions before them, are part of a long series of *realizability questions* in graph theory, having their root in the seminal work of Erdős and Gallai [14] and Havel and Hakimi [20, 18] on whether (and how, respectively) a given degree sequence can be realized as a (static) graph.

In this work, we return to the roots of realizability questions, studying the temporal analog of Erdős and Gallai's problem, namely, whether (and how) a given degree sequence can be realized as a temporal graph in TC, for both the simple and non-simple cases. This question is partly motivated by the hardness results of Göbel et al. [16], a natural question being how to relax the input specification in a way that offers more flexibility (and tractability) for realization. Another motivation is to investigate whether this format allows for an elegant and concise characterization of all the feasible cases, a situation that is pretty rare in the literature on temporal graphs.

Main results. Before stating the results, we recall that a *degree sequence* $\mathbf{d}$ is a non-increasing list of integers. A degree sequence of length n is *graphical* if there exist an n-vertex (static) graph whose vertices have exactly these values as their degrees. Moreover, $\mathbf{d}$ is *multigraphical* if a multigraph with this degree distribution exists.

Our main result is a complete characterization of the graphical sequences that admit a realization as a simple and proper temporal graph in TC. We refer to these sequences as *TC-realizable graphical sequences*. The characterization, obtained in Section 5, is the following:

▶ **Theorem 5.1.** *A graphical sequence $\mathbf{d} = (d_1, \ldots, d_n)$ with $m := \frac{1}{2} \cdot \sum_{i=1}^{n} d_i$ edges is TC-realizable if and only if one of the following holds:*

- $m = 2n - 4$, $d_1 < n - 1$, *and* $d_n \geq 2$
- $m \geq 2n - 3$ *and either (i)* $n \leq 2$ *or (ii)* $n > 2$, $d_{n-1} \geq 2$, *and* $d_n \geq 1$.

We also characterize the degree sequences that can be realized as a non-simple and proper temporal graph in TC, where *non-simple* means that a same edge can receive multiple

time labels. Conveniently, non-simple temporal graphs are equivalent to simple temporal graphs whose underlying graph is a multigraph and where each edge of the multigraph only receives one label. Thus, the problem amounts to characterizing *TC-realizable multigraphical sequences*. The characterization is:

▶ **Theorem 5.2.** *A multigraphical sequence* $\mathbf{d} = (d_1, \ldots, d_n)$ *with* $m := \frac{1}{2} \cdot \sum_{i=1}^{n} d_i$ *edges is TC-realizable if and only if one of the following holds:*

- $m = 2n - 4$ *and* $d_n \geq 2$
- $m \geq 2n - 3$ *and either (i)* $n \leq 2$ *or (ii)* $n > 2$*,* $d_{n-1} \geq 2$*, and* $d_n \geq 1$*.*

As one can see, both characterizations are close to each other, the set of TC-realizable multigraphical sequences being of course more general (indeed, TC-realizable graphical sequences are special cases of these). However, the difference is more important than one may think by looking only at these theorems, because multigraphical sequences already differ from graphical sequences. (Both standard characterizations will be recalled in the paper.)

On the algorithmic side, we show that TC-realizable graphical sequences can be recognized in $\mathcal{O}(n)$ time and TC-realizable multigraphical sequences in $\mathcal{O}(n + m)$ time. We also give a constructive $\mathcal{O}(n + m)$ time algorithm that outputs a temporal graph satisfying the desired constraints (if one exists), this algorithm being obviously asymptotically optimal.

The fact that we consider proper labelings is not a limitation. Precisely, if non-proper labelings are considered, then one may either consider strict (i.e. increasing) or non-strict (i.e. non-decreasing) temporal paths. In the case of strict temporal paths, we know that non-proper labelings can always be turned into proper labelings without reducing the reachability relation [7]. Together with the fact that proper labelings are particular cases of non-proper labelings, this implies that the feasible cases are the same as in the proper setting. If one considers non-strict temporal paths instead, then the problem becomes trivial to solve: a single spanning tree suffices (with the same time label on every edge). Thus, in this case, the problem reduces to testing if the sequence can be realized as a connected graph, which from earlier works [25, 23, 15] is the case if and only if $m \geq n - 1$ and $d_n \geq 1$.

Technical overview. The characterization of TC-realizable graphs in terms of input graphs is complicated. Some necessary and sufficient conditions are known from the work of Göbel et al. [16] (itself based on earlier works in gossip theory [19, 17, 6]), the remaining cases being NP-hard to decide. A necessary condition for these graphs is to contain two spanning trees that share at most two edges. If the graphs admit two spanning trees that share at most one edge, then this is sufficient: the graph is TC-realizable. The case with two shared edges is further constrained. If the number of edges is exactly $2n - 4$, then it is necessary and sufficient that the two shared edges belong to a central cycle of length 4. Göbel et al. [16] called such graphs "C_4*-graphs*" (although they do not consist only of a cycle of length 4. To avoid this confusion, we use the term C_4*-pivotable* graphs instead. A formal definition is given in Section 2). Otherwise, there may exist feasible cases which are difficult to characterize, and these cases are precisely the ones making the problem NP-hard. Interestingly, our results imply that the framework of degree sequences allows one to circumvent these hard cases entirely.

Our characterization proceeds as follows: First, we note that the characterization of degree sequences which are realizable as a graph admitting two edge-disjoint spanning trees is known [25]. Thus, we focus on characterizing the degree sequences that can be realized as a graph admitting two spanning trees sharing exactly one edge, and, for degree sequences corresponding to exactly $2n - 4$ edges, we characterize the ones admitting a realization with the above C_4-pivotable graph property. Surprisingly, these cases already cover all the

TC-realizable graphical sequences. Namely, if a graph has strictly more than $2n-4$ edges and is TC-realizable, then its degree sequence could *also* be realized as a graph with two spanning trees sharing at most one edge, falling back on the previous cases. In other words, a significant by-product of our analysis is that the difficult graphs can safely be ignored in the framework of degree sequences.

In light of these explanations, the feasible cases stated in Theorem 5.1 correspond respectively to the following theorems in the paper: A degree sequence is TC-realizable if and only if it admits (i) a realization which is a C_4-pivotable graph (Theorem 4.1) or (ii) a realization with two spanning trees that share at most one edge (Theorem 3.1). For multigraphical sequences, we prove the characterization similarly.

All of our proofs are constructive and rely on deconstructing recursively the degree sequence to a smaller degree sequence, using gadgets that preserve two edge-disjoint spanning trees in the constructed graph (up to a central component). These constructive proofs can be implemented efficiently by using a suitable data structure that stores the degree sequence and the respective graph efficiently. The labeling is then handled separately by a dedicated procedure that achieves the claimed time using our data structure and extra features offered by the above structural algorithms.

Temporal graphs are notoriously intractable objects. Most of the results in this young field are negative and most of the problems turn out to be computationally hard, often due to the non-symmetric and non-transitive nature of reachability. In this respect, the fact the feasible cases for TC-realizability admit a characterization that is at the same time compact, purely structural, and easy to recognize appears to be significant. This situation is clearly an exception in the landscape of temporal graphs studies.

2 Definitions and Important Existing Results

A temporal graph is a pair $\mathcal{G} = (G, \lambda)$, where $G = (V, E)$ is a standard (in this work, *undirected*) graph called the *underlying graph* of $\mathcal{G}$, and $\lambda : E \to 2^{\mathbb{N}}$ is a labeling function that assigns a non-empty set of time labels to every edge of E, interpreted as availability times. The labeling function λ can be restricted in various ways. It is called *proper* if adjacent edges cannot share a common time label (λ is locally injective), and it is called *simple* if every edge has exactly one time label (λ is single-valued). The typical setting of gossiping, including that of Göbel et al. [16], requires that the labeling is both proper and simple.

A pair (e, t) such that $t \in \lambda(e)$ is a *temporal edge* of $\mathcal{G}$. A *temporal path* in $\mathcal{G}$ is a sequence of temporal edges $\langle (e_i, t_i) \rangle$ such that $\langle e_i \rangle$ is a path in the underlying graph and $\langle t_i \rangle$ is non-decreasing (such a path is *strict* if $\langle t_i \rangle$ is increasing). A temporal graph $\mathcal{G}$ is *temporally connected* (in class TC) if temporal paths exist between all ordered pairs of nodes. Observe that, beyond modeling mutually exclusive interaction, the proper setting has the technical advantage of removing the distinction between strict and non-strict temporal paths (indeed, all the temporal paths in this case are *de facto* strict), which allows us to rely on a single definition of TC throughout the paper.

We can now state the gossiping problem as follows: We are given an undirected (multi)graph G and the question is whether there exists a simple and proper labeling of G such that the resulting temporal graph is temporally connected (i.e., in TC).

The original realizability question of Erdős and Gallai asks the following question:

Degree Sequence Realization
Input: A degree sequence $\mathbf{d} = (d_1, \ldots, d_n)$.
Question: Does there exist a graph $G = (V, E)$ with $V = \{v_1, \ldots, v_n\}$ such that v_i has degree d_i?

When the answer is yes, the degree sequence $\mathbf{d}$ is called *graphical* and G is called a *realization* of $\mathbf{d}$. If multiedges are permitted in G and if there is a multigraph that realizes $\mathbf{d}$, then the sequence is instead called *multigraphical* [9]. In this work, we study the following temporal version of the problem:

Degree Sequence TC Realization
Input: A degree sequence $\mathbf{d} = (d_1, \dots, d_n)$.
Question: Does there exist a realization $G = (V, E)$ of $\mathbf{d}$ and a labeling λ such that $(G, \lambda) \in \mathsf{TC}$?

Next, we state important existing results on (multi)graphical degree sequences and TC-realizable graphs and derive immediate consequences that we use throughout the paper.

Graphical Degree Sequences.

▶ **Theorem 2.1** (Graphical Characterization [14, 29]). *A degree sequence $\mathbf{d} = (d_1, \dots, d_n)$ is graphical if and only if*

(i) $\sum_{i=1}^{n} d_i$ *is even, and* **(ii)** *for each* $r \in [1, n-1]$: $\sum_{i=1}^{r} d_i \leq r \cdot (r-1) + \sum_{i=r+1}^{n} \min(r, d_i)$.

Based on this characterization, we obtain the following sufficient condition under which degree sequences with few edges are graphical.

▶ **Corollary 2.2** (⋆). *Let $\mathbf{d} = (d_1, \dots, d_n)$ be a degree sequence with $\sum_{i=1}^{n} d_i$ being even, $\sum_{i=1}^{n} d_i \leq 4(n-1)$, $n > 4$, $d_1 \leq n-1$, and $d_n \geq 2$. Then $\mathbf{d}$ is graphical if $d_4 \geq 3$.*

We will make use of this corollary several times in this work to easily argue that specific degree sequences are graphical. Similarly, we also obtain the following sufficient condition.

▶ **Corollary 2.3.** *Let $\mathbf{d} = (d_1, \dots, d_n)$ be a degree sequence with $\sum_{i=1}^{n} d_i = 4(n-1) - 2$, $n > 4$, $d_1 \leq n-1$, $d_{n-1} \geq 2$, and $d_n \geq 1$. Then $\mathbf{d}$ is graphical if $d_4 \geq 3$.*

The following algorithm constructs a graph from a graphical degree sequence.

▶ **Definition 2.4** (Graphical Laying Off Process [18]). *Let $\mathbf{d} = (d_1, \dots, d_n)$ be a graphical sequence. The* laying off *procedure consists of connecting the vertex v_i to the first d_i vertices, excluding v_i. The resulting* residual sequence *is given by:*

$$\begin{cases} (d_1 - 1, \dots, d_{d_i} - 1, d_{d_i+1}, \dots, d_{i-1}, d_{i+1}, \dots, d_n) & \text{if } d_i < i, \\ (d_1 - 1, \dots, d_{i-1} - 1, d_{i+1} - 1, \dots, d_{d_i+1} - 1, d_{d_i+2}, \dots, d_n) & \text{if } d_i \geq i. \end{cases}$$

The next lemma implies that the laying off process can be used iteratively to build a realization for a graphical degree sequence.

▶ **Lemma 2.5** ([13, 18]). *If $\mathbf{d} = (d_1, \dots, d_n)$ is a graphical sequence, then the residual sequence after laying off any entry is also graphical.*

Multigraphical degree sequences.

▶ **Theorem 2.6** (Multigraphical Characterization [18]). *A degree sequence $\mathbf{d} = (d_1, \dots, d_n)$ is multigraphical if and only if*

(i) $\sum_{i=1}^{n} d_i$ *is even, and* **(ii)** $d_1 \leq \sum_{i=2}^{n} d_i$.

For multigraphical sequences there also exists a "laying off" process which is similar to Definition 2.4, but more flexible. Contrary to the graphical case, where a whole degree is laid off, in the multigraphical case, only a single edge is laid off.

▶ **Theorem 2.7** (Multigraphical Laying Off Process [4]). *Let $\mathbf{d} = (d_1, \ldots, d_n)$ be a multigraphical degree sequence with $d_n > 0$. Then, for each $j, 2 \leq j \leq n$, the degree sequence $(d_1 - 1, d_2, \ldots, d_{j-1}, d_j - 1, d_{j+1}, \ldots, d_n)$ is multigraphical (after reordering).*

(Multi)graphical sequences admitting two edge-disjoint spanning trees. The characterization of sequences admitting a realization with k edge-disjoint spanning trees is identical for graphical and multigraphical sequences. For graphical sequences, it was proven constructively by Kundu for $k = 2$ [25] and extended to general k by Kleitman and Wang [23]. The multigraphical case was shown non-constructively by Gu, Hai, and Liang [15]. While we focus on the case $k = 2$, we state the full characterization below.

▶ **Theorem 2.8** ([25, 23, 15]). *A (multi)graphical sequence $\mathbf{d} = (d_1, \ldots, d_n)$ with $n \geq 2$ admits a (multigraphical) realization with $k \in \mathbb{N}$ edge-disjoint spanning trees if and only if* **(i)** $d_n \geq k$, *and* **(ii)** $\sum_{i=1}^{n} d_i \geq 2 \cdot k(n-1)$.

TC-realizable graphs. Finally, we provide an overview on some known necessary and sufficient conditions of TC-realizable (multi)graphs.

▶ **Lemma 2.9** ([17]). *Let G be a TC-realizable (multi)graph on n vertices. Then G has at least $2n - 4$ edges.*

▶ **Lemma 2.10** ([3]). *Let G be a (multi)graph with two spanning trees that share at most one edge. Then G is TC-realizable.*

▶ **Definition 2.11** (Reformulation of Göbel et al. [16]). *A graph G on n vertices and $2n - 4$ edges is a C_4-*pivotable graph *if G contains an induced cycle C of length 4 (called a* central cycle*) and two spanning trees that share exactly two edges, where both shared edges are from C.*

This definition is equivalent to the one provided by Göbel et al. [16], since no tree can contain more than three edges of the central cycle, and due to the number of edges, each edge of the cycle is contained in at least one of the two spanning trees.

▶ **Lemma 2.12** ([16]). *Let G be a graph with n vertices and $2n - 4$ edges. Then G is TC-realizable if and only if G is a C_4-pivotable graph.*

Even though it is not explicitly stated [16], the following generalization of C_4-pivotable graphs to multigraphs also is TC-realizable by the same labeling procedure used for C_4-pivotable graphs.[1]

▶ **Definition 2.13.** *A multigraph G on n vertices and $2n-4$ edges is a C_4-*pivotable multigraph *if G contains an induced cycle C of length 4 (called a* central cycle*) and two spanning trees that share exactly two edges, where both shared edges are from C.*

Here, an induced cycle in a multigraph G has the additional property that each edge of C exists exactly once in G. Note that a C_4-pivotable graph is also a C_4-pivotable multigraph.

▶ **Corollary 2.14.** *Let G be a C_4-pivotable multigraph. Then G is TC-realizable.*

[1] We will recall this labeling procedure in the full version.

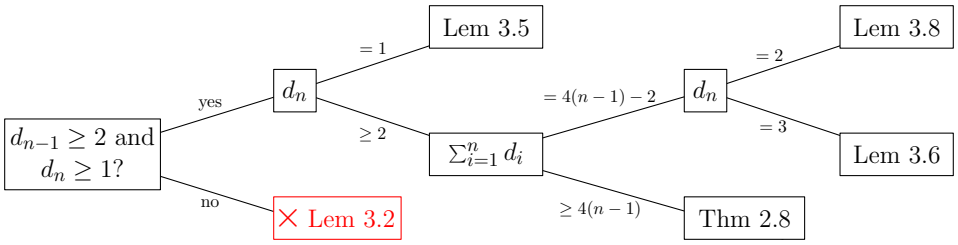


Figure 1 A guideline for the individual steps for the proof of Theorem 3.1.

3 Realizations with Two Spanning Trees Sharing at Most one Edge

In this section, we establish the following characterization of sequences that allow for a realization with two spanning trees that share (at most) one edge.

▶ **Theorem 3.1.** *Let* $\mathbf{d} = (d_1, \ldots, d_n)$ *be a graphical sequence. Then,* $\mathbf{d}$ *admits a realization with two spanning trees that share at most one edge if and only if* $\sum_{i=1}^{n} d_i \geq 4(n-1)-2$ *and (i)* $n \leq 2$ *or (ii)* $n > 2$*,* $d_{n-1} \geq 2$*, and* $d_n \geq 1$*.*

First, we consider the special cases of $n \leq 2$. To this end, note that $\{(0), (0,0), (1,1)\}$ are exactly the graphical sequences of length at most 2. Each of these sequences has a unique realization and $(0,0)$ is the only sequence for which the unique realization is not connected. Hence for $\mathbf{d} = (0,0)$, there is no realization with two spanning trees that share at most one edge and $\sum \mathbf{d} = 0 < 2 = 4(n-1)-2$. For $\mathbf{d} \in \{(0), (1,1)\}$, $\sum \mathbf{d} \geq 4(n-1)-2$ and there is no realization with two spanning trees that share at most one edge. This proves the equivalence for part (i) of Theorem 3.1.

By the above, we may assume $n > 2$ and proceed to show the forward direction for part (ii) of Theorem 3.1. Figure 1 provides a guideline for the steps of the proof of this case.

▶ **Lemma 3.2** (⋆)**.** *Let* $\mathbf{d} = (d_1, \ldots, d_n)$ *be a graphical sequence with* $n > 2$*. If* $\mathbf{d}$ *admits a realization with two spanning trees that share at most one edge, then* $\sum_{i=1}^{n} d_i \geq 4(n-1)-2$*,* $d_{n-1} \geq 2$*, and* $d_n \geq 1$*.*

Realizability. Now, we consider the backward direction for part (ii) of Theorem 3.1. That is, we will show the following.

▶ **Lemma 3.3.** *Let* $\mathbf{d} = (d_1, \ldots, d_n)$ *be a graphical sequence with* $\sum_{i=1}^{n} d_i \geq 4(n-1)-2$*,* $n > 2$*,* $d_{n-1} \geq 2$*, and* $d_n \geq 1$*. Then* $\mathbf{d}$ *admits a realization with two spanning trees that share at most one edge.*

First, recall Theorem 2.8, for which the following is a corollary.

▶ **Corollary 3.4** ([23])**.** *Let* $\mathbf{d} = (d_1, \ldots, d_n)$ *be a graphical sequence with* $\sum_{i=1}^{n} d_i \geq 4(n-1)$*,* $n > 2$*, and* $d_n \geq 2$*. Then* $\mathbf{d}$ *admits a realization that has two edge-disjoint spanning trees.*

Hence, it remains to consider (i) $\sum \mathbf{d} \geq 4(n-1)-2$, $d_{n-1} \geq 2$, and $d_n = 1$, and (ii) $\sum \mathbf{d} = 4(n-1)-2$ and $d_n \geq 2$. We first consider the former.

▶ **Lemma 3.5** (⋆)**.** *Let* $\mathbf{d}$ *be a graphical sequence with* $\sum_{i=1}^{n} d_i \geq 4(n-1)-2$*,* $n > 2$*,* $d_{n-1} \geq 2$*, and* $d_n = 1$*. Then* $\mathbf{d}$ *has a realization with two spanning trees that share one edge.*

Now, we consider $\sum \mathbf{d} = 4(n-1)-2$ and $d_n \geq 2$. Note that these sequences are guaranteed to fulfill $d_n \in \{2, 3\}$, as otherwise, $\sum \mathbf{d} \geq 4n > 4(n-1)-2$. We first prove the case of $d_n = 3$ in Lemma 3.6, which is needed for the subsequent case $d_n = 2$ in Lemma 3.8.

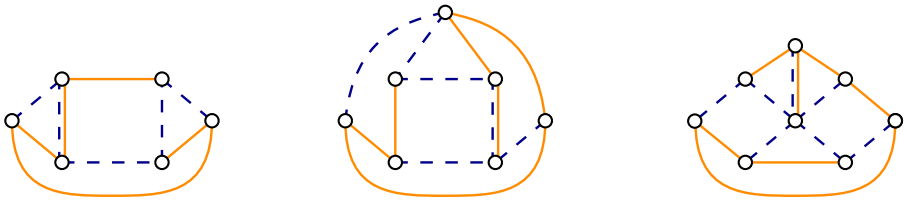


Figure 2 Realizations of the degree sequences $\mathbf{d}^6 = (3,3,3,3,3,3)$, $\mathbf{d}^7 = (4,3,3,3,3,3,3)$, and $\mathbf{d}^8 = (5,3,3,3,3,3,3,3)$ with two spanning trees T_1 (solid orange edges) and T_2 (dashed blue edges) that share exactly one edge, shown from left to right.

▶ **Lemma 3.6.** *Let $\mathbf{d} = (d_1, \ldots, d_n)$ be a graphical sequence with $\sum_{i=1}^{n} d_i = 4(n-1) - 2$ and $d_n = 3$. Then $\mathbf{d}$ admits a realization with two spanning trees that share one edge.*

Proof. We will show that $n \geq 6$ in this case. To this end, we provide a lower bound on the number of entries of value 3 in $\mathbf{d}$.

▷ Claim 3.7 (⋆). Let ℓ be the smallest index of $[1, n]$ for which $d_\ell = 3$. Then, the number of entries of value 3 in $\mathbf{d}$ is $6 + \sum_{i=1}^{\ell-1}(d_i - 4)$.

Note that this implies that $d_1 < n-1$, as otherwise, $\mathbf{d}$ would contain at least $n-1-4+6 > n$ entries of value 3. We distinguish between $d_2 = 3$ and $d_2 \geq 4$.

Firstly, consider the case that $d_2 = 3$. Note that this implies by Claim 3.7 that $d_1 = n-3$, since $\sum \mathbf{d} = 4(n-1) - 2$. In other words, for each $n \geq 6$, there is exactly one degree sequence $\mathbf{d}^n := (d_1^n, \ldots, d_n^n)$ of length n with $d_2^n = d_n^n = 3$ fulfilling $\sum \mathbf{d}^n = 4(n-1) - 2$. Realizations with two spanning trees that share one edge for the three sequences $\mathbf{d}^6$, $\mathbf{d}^7$, and $\mathbf{d}^8$ are depicted in Figure 2. Now, consider $n \geq 9$. Recall that $d_1 = n - 3$ and $d_i = 3$ for each $i \in [2, n]$. Moreover, recall that $\mathbf{d}^6 = (3,3,3,3,3,3)$ has a realization G' with two spanning trees T_1' and T_2' that share exactly one edge (see Figure 2). Let v_1 be an arbitrary vertex of degree 3 in G'. We obtain a graph G by adding a cycle C of length $d_1 - 3 = n - 6$ to G' and adding the edge v_1x to G' for each vertex x of the cycle C. Note that G is a realization of $\mathbf{d}$, since G contains $5 + d_1 - 3 = n - 1$ vertices of degree 3 and the degree of v_1 was increased by $d_1 - 3$ to d_1. Since $G^c := G[\{v_1\} \cup C]$ consists of a universal vertex attached to a cycle of length at least 3 (since $d_1 - 3 = n - 6 \geq 3$ by $n \geq 9$), G^c contains two edge-disjoint spanning trees T_1^c and T_2^c. Hence, $T_1 := T_1' \cup T_1^c$ and $T_2 := T_2' \cup T_2^c$ are spanning trees of G, and T_1 and T_2 share only one edge, namely the edge shared by T_1' and T_2'. Hence, $\mathbf{d}$ admits a realization with two spanning trees that share one edge.

Secondly, consider the case that $d_2 \geq 4$. Note that Claim 3.7 thus implies that $n \geq d_1 + 4$, since $d_2 \geq 4$ and $\mathbf{d}$ contains at least $d_1 - 4 + 6 = d_1 + 2$ entries of value 3. Let $\mathbf{d}'$ be the sequence obtained from $\mathbf{d}$ by removing the entry d_1, removing $d_1 - 1$ entries of value 3, and adding a 1. We first show that $\mathbf{d}'$ admits a realization with two spanning trees that share one edge, and then describe how to construct the realization for $\mathbf{d}$.

The remainder of this case is deferred to the full version. Intuitively, to then obtain a desired realization, we attach, similar to the the first case, a large cycle to a degree-1 vertex v_1. This time, the cycle has length $d_1 - 1$ instead of $d_1 - 3$ as in the first case. ◀

Finally, we consider the case where $\sum \mathbf{d} = 4(n-1) - 2$ and $d_n = 2$. Essentially, we perform an induction over the length of the sequence and lay off the smallest degree $d_n = 2$, until we reach a sequence of constant length or d_n has value unequal to 2. In the latter case, the previous lemma acts as a base case.

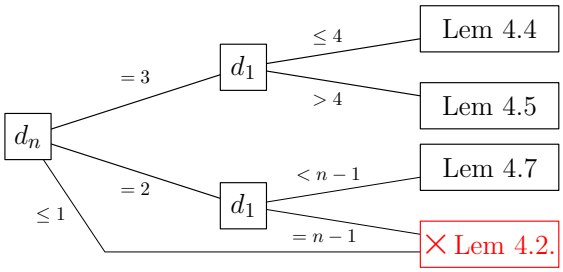


Figure 3 A guideline for the individual steps for the proof of Theorem 4.1.

▶ **Lemma 3.8** (⋆). *Let* $\mathbf{d} = (d_1, \ldots, d_n)$ *be a graphical sequence with* $\sum_{i=1}^{n} d_i = 4(n-1) - 2$ *and* $d_n = 2$*. Then* $\mathbf{d}$ *admits a realization with two spanning trees that share one edge.*

4 Realizations of C_4-pivotable Graphs

In this section, we establish the following characterization of graphical sequences that allow for a C_4-pivotable graph realization.

▶ **Theorem 4.1.** *Let* $\mathbf{d} = (d_1, \ldots, d_n)$ *be a graphical sequence. Then,* $\mathbf{d}$ *admits a realization which is a* C_4*-pivotable graph if and only if* $\sum_{i=1}^{n} d_i = 4(n-1) - 4$*,* $d_1 < n - 1$*, and* $d_n \geq 2$*.*

Figure 3 provides a guideline on the individual steps for proving Theorem 4.1. First, we show the forward direction of Theorem 4.1. To this end, we analyze the minimum and maximum degree of C_4-pivotable graphs. This then directly proves the forward direction of Theorem 4.1.

▶ **Lemma 4.2** (⋆). *Let* G *be a* C_4*-pivotable graph on* n *vertices. Then* G *has a minimum degree of at least* 2 *and a maximum degree of at most* $n - 2$*.*

Realizability. Now, we show the backward direction. That is, we show the following.

▶ **Lemma 4.3.** *Let* $\mathbf{d} = (d_1, \ldots, d_n)$ *be a graphical sequence with* $\sum_{i=1}^{n} d_i = 4(n-1) - 4$*,* $d_1 < n - 1$*, and* $d_n \geq 2$*. Then,* $\mathbf{d}$ *admits a realization which is a* C_4*-pivotable graph.*

To prove this statement, we assume that we are given a graphical sequence $\mathbf{d}$ with $\sum \mathbf{d} = 4(n-1) - 4$, $d_1 < n - 1$, and $d_n \geq 2$. Note that these sequences are guaranteed to fulfill $d_n \in \{2, 3\}$, as otherwise, $\sum \mathbf{d} \geq 4n > 4(n-1) - 4$. We split the analysis in three cases. First, we consider the case of $d_n = 3$ and $d_1 \leq 4$ in Lemma 4.4, then $d_n = 3$ and $5 \leq d_1 < n-1$ in Lemma 4.5, and lastly $d_n = 2$ and $d_1 < n - 1$ in Lemma 4.6.

▶ **Lemma 4.4.** *Let* $\mathbf{d} = (d_1, \ldots, d_n)$ *be a graphical sequence with* $\sum_{i=1}^{n} d_i = 4(n-1) - 4$*,* $d_1 \leq 4$ *and* $d_n = 3$*. Then* $\mathbf{d}$ *admits a realization which is a* C_4*-pivotable graph.*

Proof. Since $\sum \mathbf{d} = 4(n-1) - 4$, $d_1 \leq 4$, and $d_n = 3$, we get that $n \geq 8$, $d_{n-7} = 3$, and (if $n > 8$), $d_{n-8} = 4$. In other words, for each $n \geq 8$, there is exactly one degree sequence $\mathbf{d}^n$ of length n with the required properties.

We show that $\mathbf{d}^n$ admits a realization which is a C_4-pivotable graph via induction over n. In fact, we show a stronger result, namely, that for each $n \geq 8$, $\mathbf{d}^n$ has a realization G which is a C_4-pivotable graph with central cycle C, such that (i) there are two spanning trees T_1 and T_2 of G that share exactly two edges and both belong to C, and (ii) for each $i \in [1, 2]$,

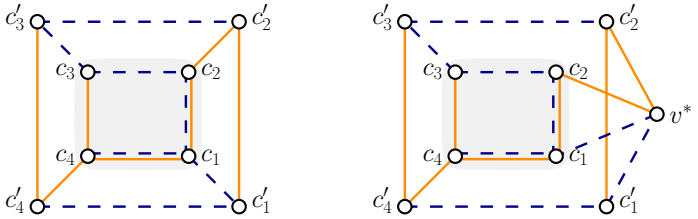


Figure 4 Realization of the degree sequence $\mathbf{d}^8 = (3,3,3,3,3,3,3,3)$ from the base case in the proof of Lemma 4.4, featuring the central 4-cycle C highlighted in gray, and two spanning trees T_1 (solid orange edges) and T_2 (dashed blue edges) that share exactly two edges, both contained in C. The right side illustrates the inductive step.

there are two distinct edges e_i, f_i from T_i and outside of C for which $\{e_1, e_2\}$ and $\{f_1, f_2\}$ are matchings in G. We call edge pairs (e_1, e_2) and (f_1, f_2) fulfilling these conditions *matching edge pairs*.

For the base case of $n = 8$, consider the graph given in Figure 4 which is a realization of $\mathbf{d}^8 = (3,3,3,3,3,3,3,3)$. As highlighted in the figure, there is a central cycle C on the vertices c_1, c_2, c_3, c_4. Moreover, $(c_1c'_1, c_2c'_2)$ and $(c_3c'_3, c_4c'_4)$ are matching edge pairs. Hence, the statement holds for the base case of $n = 8$.

Now let $n > 8$ and assume that the statement holds for $n-1$. We show that the statement also holds for n. Let G' be a realization of $\mathbf{d}^{n-1}$ according to the properties of the induction hypothesis. That is, G' is a C_4-pivotable graph with central cycle C, two spanning trees T'_1 and T'_2 that share exactly two edges, both of which belong to C, and matching edge pairs (e'_1, e'_2) and (f'_1, f'_2). Recall that $\mathbf{d}^n$ can be obtained from $\mathbf{d}^{n-1}$ by adding a single entry of value 4. The idea to obtain a realization for $\mathbf{d}^n$ with these properties is by essentially subdividing both edges $e'_1 = pq$ and $e'_2 = xy$ and identifying both newly added degree-2 vertices to obtain a new vertex v^* (which then has degree 4). Formally, we obtain a graph G by removing the edges e'_1 and e'_2 from G' and afterwards adding a new vertex v^* together with the edges pv^*, qv^*, xv^*, yv^*.

Note that each vertex of $V(G) \setminus \{v^*\}$ has the same degree in both G and G' and v^* has degree 4. Hence, G is a realization of $\mathbf{d}^n$. Moreover, since e'_1 and e'_2 are not part of the central cycle C of G', C is still a central cycle where $T_1 := (T'_1 - e'_1) \cup \{pv^*, qv^*\}$ and $T_2 := (T'_2 - e'_2) \cup \{xv^*, yv^*\}$ being spanning trees of G that each share exactly two edges and both of which belong to C. This implies that G is a C_4-pivotable graph. It thus remains to show that there are matching edge pairs (e_1, e_2) and (f_1, f_2) in G. To define these pairs, we use the so far undiscussed pair (f'_1, f'_2). Since e'_1, e'_2, f'_1, f'_2 are pairwise distinct edges in G', f'_1 and f'_2 are still edges of G and moreover, f'_1 is an edge of T_1 and f'_2 is an edge of T_2. Recall that xv^* and yv^* are edges of T_2, and note that f'_1 contains at most one of x and y as an endpoint, as otherwise f'_1 would be identical to e'_2. Hence, there is an edge $e_2 \in \{xv^*, yv^*\}$ which is not adjacent to f'_1, that is, $\{f'_1, e_2\}$ is a matching in G. With the same arguments, there is an edge $e_1 \in \{pv^*, qv^*\}$ which is not adjacent to f'_2, that is, $\{e_1, f'_2\}$ is a matching in G. This implies that (f'_1, e_2) and (e_1, f'_2) are matching edge pairs. Consequently, the statement holds for n.

Concluding, for each $n \geq 8$, the degree sequence $\mathbf{d}^n$ admits a C_4-pivotable graph realization. ◀

Next, we consider the case that $5 \leq d_1 < n-1$ and $d_n = 3$.

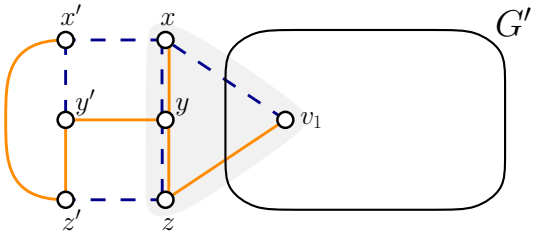


Figure 5 Construction of the graph G described in the proof of Lemma 4.5.

▶ **Lemma 4.5** (⋆). *Let* $\mathbf{d} = (d_1, \ldots, d_n)$ *be a graphical sequence with* $\sum_{i=1}^{n} d_i = 4(n-1) - 4$, $5 \leq d_1 < n - 1$, *and* $d_n = 3$. *Then* $\mathbf{d}$ *admits a realization which is a* C_4*-pivotable graph.*

Sketch. Essentially, we can show that the degree sequence $\mathbf{d}'$ obtained by reducing v_1 by 2 and removing six entries of value 3, has a realization G' with two edge-disjoint spanning trees. To this graph, we then attach the six vertices as depicted in Figure 5 to a vertex v_1 of degree $d_1 - 2$ in G'. The resulting graph is then a C_4-pivotable graph realization of $\mathbf{d}$. ◀

Finally, we consider the case where $d_1 < n - 1$ and $d_n = 2$. This proof is similar to the proof of Lemma 3.8. Essentially, we perform an induction over the length of the sequence and lay off the smallest degree $d_n = 2$, until we reach a sequence of constant length or d_n has value 3. In the latter case, one of the previous two lemmas acts as a base case.

▶ **Lemma 4.6** (⋆). *Let* $\mathbf{d} = (d_1, \ldots, d_n)$ *be a graphical sequence with* $\sum_{i=1}^{n} d_i = 4(n-1) - 4$, $d_1 < n - 1$, *and* $d_n = 2$. *Then* $\mathbf{d}$ *admits a realization which is a* C_4*-pivotable graph.*

5 The Characterizations for TC-realizable Sequences

We now prove our characterization for graphical and multigraphical TC-realizable sequences.

5.1 Realizations of TC-realizable Graphs

We now prove our characterization of TC-realizable graphical sequences restated as follows.

▶ **Theorem 5.1.** *A graphical sequence* $\mathbf{d} = (d_1, \ldots, d_n)$ *with* $s = \sum_{i=1}^{n} d_i$ *is TC-realizable if and only if one of the following holds*

- $s = 4(n-1) - 4$, $d_1 < n - 1$, *and* $d_n \geq 2$
- $s \geq 4(n-1) - 2$ *and (i)* $n \leq 2$ *(ii)* $n > 2$, $d_{n-1} \geq 2$, *and* $d_n \geq 1$.

Proof. ($\Leftarrow$) If $\sum \mathbf{d} = 4(n-1) - 4$, $d_1 < n - 1$, and $d_n \geq 2$, then due to Theorem 4.1, $\mathbf{d}$ has a realization G which is a C_4-pivotable graph. By Harary and Schwenk [19] this implies that G is TC-realizable and thus that $\mathbf{d}$ is TC-realizable.

If $\sum \mathbf{d} \geq 4(n-1) - 2$ and (i) $n \leq 2$ or (ii) $n > 2$, $d_{n-1} \geq 2$, and $d_n \geq 1$, then $\mathbf{d}$ admits a realization G which has two spanning trees that share at most one edge (see Theorem 3.1). By Harary and Schwenk [19] this implies that G is TC-realizable and thus that $\mathbf{d}$ is TC-realizable.

($\Rightarrow$) Suppose that $\mathbf{d}$ is TC-realizable and let G be one of its realizations. By Hajnal et al. [17], G has at least $2n - 4$ edges, which implies $\sum \mathbf{d} \geq 4(n-1) - 4$.

First, consider the case that $\sum \mathbf{d} = 4(n-1) - 4$. That is, G has $2n - 4$ edges. Hence, by Göbel et al. [16], G is a C_4-pivotable graph. Thus, by Theorem 4.1, $d_1 < n - 1$, and $d_n \geq 2$.

Second, consider the case that $\sum \mathbf{d} \geq 4(n-1) - 2$. Since G is TC-realizable, G is connected. For $n \leq 2$, $\mathbf{d}$ is from $\{(0), (1,1)\}$ as these are the only graphical sequences of length at most 2 fulfilling $\sum \mathbf{d} \geq 4(n-1) - 2$. Thus, consider $n > 2$. Since G is connected

and $n > 2$, G has a minimum degree of 1, which implies that $d_n \geq 1$. It remains to show that $d_{n-1} \geq 2$. To this end, assume towards a contradiction that $d_{n-1} = d_n = 1$. Then, G contains at least two vertices u and v of degree 1. These vertices are not adjacent in G, as otherwise, G would not be connected. Hence, the unique edge e_u incident with u and the unique edge e_v incident with v are distinct. Since G is TC-realizable, there is a TC-labeling of the edges of G. To ensure that there is a temporal path from u to v, this labeling has to assign a label to e_u which is strictly smaller than the label assigned to e_v. But simultaneously, to ensure that there is a temporal path from v to u, the label assigned to e_v has to be strictly smaller than the label assigned to e_u; a contradiction. Consequently, G contains no two vertices of degree 1, which implies that $d_{n-1} \geq 2$ and $d_n \geq 1$ in the case that $n > 2$. ◀

5.2 Realizations of TC-realizable Multigraphs

We now prove our characterization for TC-realizable multigraphical sequences:

▶ **Theorem 5.2** (⋆). *A multigraphical sequence* $\mathbf{d} = (d_1, \ldots, d_n)$ *with* $s := \sum_{i=1}^{n} d_i$ *is TC-realizable if and only if one of the following holds*

- $s = 4(n-1) - 4$ *and* $d_n \geq 2$
- $s \geq 4(n-1) - 2$ *and (i)* $n \leq 2$ *or (ii)* $n > 2$, $d_{n-1} \geq 2$, *and* $d_n \geq 1$.

Note that the only difference from the characterization of (simple) graphical sequences (see Theorem 5.1) is the absence of the restriction $d_1 < n - 1$ in the case $m = 4(n-1) - 4$.

We show the statement in the following similarly to the graphical case.

Intermediate characterization for spanning trees with one shared edge. First, we provide a characterization for multigraphical sequences that allow for a realization that has two spanning trees with at most one shared edge.

▶ **Theorem 5.3** (⋆). *Let* $\mathbf{d} = (d_1, \ldots, d_n)$ *be a multigraphical sequence. Then,* $\mathbf{d}$ *admits a realization with two spanning trees that share at most one edge if and only if* $\sum_{i=1}^{n} d_i \geq 4(n-1) - 2$ *and (i)* $n \leq 2$ *or (ii)* $n > 2$, $d_{n-1} \geq 2$, *and* $d_n \geq 1$.

Intermediate characterization for C_4-pivotable multigraphs. Next, we provide a characterization for multigraphical sequences that allow for a realization which is a C_4-pivotable multigraph.

▶ **Theorem 5.4.** *Let* $\mathbf{d} = (d_1, \ldots, d_n)$ *be a multigraphical sequence. Then,* $\mathbf{d}$ *admits a realization which is a* C_4*-pivotable multigraph if and only if* $\sum_{i=1}^{n} d_i = 4(n-1) - 4$ *and* $d_n \geq 2$.

Proof. ($\Rightarrow$) By definition, each C_4-pivotable multigraph on n vertices has exactly $2n - 4$ multiedges. Hence, $\sum \mathbf{d} = 4(n-1) - 4$ holds. Next, we show that $d_n \geq 2$ also has to hold. The argument is identical to the one in the proof of Lemma 4.2: By definition, each vertex v of a the C_4-pivotable multigraphs G is (i) part of the central cycle or (ii) a vertex that has at least one incident multiedge in each of the two spanning trees that only share edges of the central cycle. In both cases, vertex v has degree at least 2. Hence, the minimum degree of G is at least 2, which implies that $d_n \geq 2$ to allow for a realization which is a C_4-pivotable multigraph. This completes this direction.

($\Leftarrow$) We have to show that $\mathbf{d}$ has a realization which is a C_4-pivotable multigraph if $\sum \mathbf{d} = 4(n-1) - 4$ and $d_n \geq 2$. Note that this implies that $n \geq 4$. To show the statement, we distinguish between the possible values of d_{n-2}.

If $d_{n-2} \geq 3$, then $\sum \mathbf{d} - d_1 \geq 3(n-3) + d_{n-1} + d_n \geq 3(n-3) + 4 = 3(n-1) - 2$. By $\sum \mathbf{d} = 4(n-1) - 4$, this implies $d_1 \leq n - 3$. Thus, by Corollary 2.2, $\mathbf{d}$ is also a graphical sequence, since $d_4 \geq 3$. This is due to the fact that $d_{n-2} \geq 3$ and $n \geq 8$, as otherwise, $\sum \mathbf{d} \geq 3n > 4(n-1) - 4$. Hence, Theorem 4.1 thus implies that $\mathbf{d}$ admits a realization which is a C_4-pivotable graph (and thus a C_4-pivotable multigraph).

Hence, we only need to consider the case that $d_{n-2} = 2 = d_{n-1} = d_n$. First, we consider special cases: $\mathbf{d} \in \{(2,2,2,2), (3,3,2,2,2)\}$. Note that these are the only two options for $\mathbf{d}$ for which $\sum \mathbf{d} = 4(n-1) - 4$, $d_1 \leq 4$, and $d_{n-2} = d_{n-1} = d_n = 2$. Both these degree sequences are graphical and fulfill $d_1 < n - 1$ and $d_n \geq 2$. Hence, by Theorem 4.1, $\mathbf{d}$ admits a realization which is a C_4-pivotable graph (and thus a C_4-pivotable multigraph).

For all other options of $\mathbf{d}$, we can thus assume that $d_1 \geq 4$. Let $\mathbf{d}' := (d'_1, \ldots, d'_{n'})$ be the degree sequence with $n' := n - 3$ obtained from $\mathbf{d}$ by (i) removing d_{n-2}, d_{n-1}, and d_n, and by (ii) reducing d_1 by 2 (and potentially reordering).

▷ Claim 5.5 (⋆). $\mathbf{d}'$ admits a realization G' with two edge-disjoint spanning trees T'_1 and T'_2.

We obtain a realization G of $\mathbf{d}$ as follows: Let v_1 denote an arbitrary vertex of G' with degree $d_1 - 2$. We add three new vertices x, y, and z to G' and add the edges xy, yz, v_1x, v_1z. Note that G is a realization of $\mathbf{d}$, since we added three new vertices of degree 2 to G' and increased the degree of a vertex of degree $d_1 - 2$ by 2. It remains to show that G is a C_4-pivotable multigraph. This is the case, since $C := (v_1, x, y, z)$ is an induced cycle of length 4 in G and $T_1 := T'_1 \cup \{v_1x, xy, yz\}$ and $T_2 := T'_2 \cup \{v_1z, xy, yz\}$ are two spanning trees of G that only share two edges and both these edges are from C (namely the edges xy and yz). Consequently, $\mathbf{d}$ admits a realization which is a C_4-pivotable multigraph. ◀

With these characterizations, we can prove our characterization for TC-realizable multigraphical sequences. The proof is similar to Theorem 5.1 and deferred to the full version.

6 Linear-Time Algorithm for Realizing TC (Multi)graphs

In the previous section, we presented a full characterization of (multi)graphical degree sequences that are TC-realizable (see Theorems 5.1 and 5.2). These necessary and sufficient conditions can be checked in $\mathcal{O}(n)$ time in the graphical and in $\mathcal{O}(n + m)$ time in the multigraphical case, where $m := \frac{1}{2} \cdot \sum \mathbf{d}$ denotes the number of edges in each realization. Hence, in linear time, we can detect whether there is a TC-realizable graph for the given degree sequence. It is, however, not clear how efficiently we can build such a realization. In this section, we show that in both the graphical and multigraphical case, we can construct such a realization (together with a TC-labeling) in $\mathcal{O}(n + m)$ time. To this end, we observe that all existential proofs in the previous sections are constructive and allow for linear time implementations with the help of a simple data structure.

First, we describe the data structure and its properties, afterwards, we explain how all previous constructive algorithms can be implemented in linear time, and finally, we describe how to obtain a TC-labeling for the constructed realization in linear time.

The data structure. We assume that all graphs be stored via adjacency list. Since our arguments rely on spanning trees, we assign two Boolean flags to each edge to indicate whether this edge is part of the first tree, the second tree, both, or neither of them. This will allow to extract the two spanning trees in $\mathcal{O}(n + m)$ time, once the final realization is constructed.

We now come to the additional data structure for degree sequences and (multi)graphs. Instead of storing degree sequences as sequences of numbers, we store them as double-linked list that store entries of the form (x, c). Such an entry indicates that the respective degree sequence contains exactly c entries of value x. The entries of the double-linked list are sorted decreasingly according to their x-values. In this way, both in the graphical and multigraphical case, we can lay off a degree d_i in $\mathcal{O}(d_i)$ time, since we only need to remove d_i and rearrange the c-values of the at most $d_i + 1$ first entries of the data structure. In this data structure, we can ensure that the degree sequence always stays sorted and no reordering is necessary.

Similarly, we use an additional data structure for (multi)graphs G. It is also a double-linked list and stores entries of the form (x, S). Such an entry indicates that the vertices of S are exactly the vertices of G that have degree exactly x. These entries are also sorted decreasingly according to their x-values. This way, we can reattach a vertex of degree d_n to G by adding edges properly to vertices of the first d_n entries of the data structure. That is, we can obtain a realization for the degree sequence that, after laying off a vertex of degree d_n, leads to the degree sequence of G. In particular, this can be done in $\mathcal{O}(d_n)$ time.

In case of a realization for a C_4-pivotable (multi)graph, we also output the central cycle separately.

Realizations for two edge-disjoint spanning trees. We now turn to the algorithmic aspects of finding a realization with two edge-disjoint spanning trees and show that such a realization can be computed in linear time.

For the graphical case, we used the result by Kundu [25]. While their proof is constructive in nature, the arguments are brief and incomplete. Thus, we give a complete construction in Lemma 6.1 and argue how it can be implemented in $\mathcal{O}(n + m)$ time using our data structure.

For the multigraphical case, we build on the result by Gu et al. [15], which was proven non-constructively. In Lemma 6.4, we give a complete inductive construction for such multigraphical sequences, and argue that this construction, too, can be carried out in $\mathcal{O}(n + m)$ time using our data structure.

▶ **Lemma 6.1 (⋆).** *Let $\mathbf{d} = (d_1, \dots, d_n)$ be a graphical sequence with $\sum \mathbf{d} \geq 4(n-1)$ and $d_n \geq 2$. We can compute a realization G of $\mathbf{d}$ and two disjoint spanning trees of G in $\mathcal{O}(n + m)$ time.*

Proof. To prove the statement, we show that for each n and for each graphical sequence $\mathbf{d}$ of length n with $\sum \mathbf{d} \geq 4(n-1)$ and $d_n \geq 2$, we can construct a realization with two edge-disjoint spanning trees. We show this statement via induction over n.

For the base case, note that for $n \in \{1, 2, 3\}$, there is no graphical sequence of length n that sums up to $4(n-1)$. Moreover, for $n = 4$, $(3, 3, 3, 3)$ is the only graphical sequence of length n with $\sum \mathbf{d} \geq 4(n-1)$ and $d_n \geq 2$. Since the complete graph on four vertices, which obviously contains two edge-disjoint spanning trees, is the only realization of $\mathbf{d}$, the base case holds for $n \leq 4$.

For the inductive step, let $n \geq 5$ and assume that the statement holds for $n-1$. Let $\mathbf{d}$ be an arbitrary graphical sequence of length n fulfilling $\sum \mathbf{d} \geq 4(n-1)$ and $d_n \geq 2$. We show that $\mathbf{d}$ admits a realization with two edge-disjoint spanning trees. Observe that whenever $\sum \mathbf{d} = 4(n-1)$, it holds that $d_n \in \{2, 3\}$ as otherwise $\sum \mathbf{d} \geq 4n$.

If $\sum \mathbf{d} = 4(n-1)$ and $d_n = 2$, or $\sum \mathbf{d} > 4(n-1)$ and $d_n \geq 2$, consider the residual sequence $\mathbf{d}' = (d'_1, \dots, d'_{n'})$ of length $n' = n - 1$ obtained from laying off the entry d_n according to Definition 2.4. In the full version, we show that this degree sequence fulfills the conditions of the induction hypothesis and thus admits a realization with two edge-disjoint spanning trees. Moreover, we can then obtain a desired realization for $\mathbf{d}$ by reattaching a vertex of degree d_n.

Otherwise, $\sum \mathbf{d} = 4(n-1)$ and $d_n = 3$. In that case, consider the sequence $\mathbf{d}'$ obtained from $\mathbf{d}$ by reducing d_1 by 1 and removing the entry d_n.

▷ Claim 6.2 (⋆). $\mathbf{d}'$ admits a realization G' with two edge-disjoint spanning trees T_1' and T_2'.

Now consider the graph G obtained from G' by adding a degree-3 vertex v_n, connecting it to a vertex v_1 of G' of degree $d_1 - 1$, and inserting it on an arbitrary edge v_iv_j of G' with $v_1 \notin \{v_i, v_j\}$ (replacing v_iv_j with v_nv_i and v_nv_j). Then G is a realization of $\mathbf{d}$. Moreover, G contains the edge-disjoint spanning trees $T_1 := T_1' \cup v_1v_n$ and $T_2 := (T_2' - v_iv_j) \cup \{v_nv_i, v_nv_j\}$.

We defer the analysis of the running time to the full version. ◀

For multigraphical sequences, we split the proof in two parts: we first show how to construct the desired realization for a multigraphical sequence $\mathbf{d}$ with $\sum \mathbf{d} = 4(n-1)$ and $d_n \geq 2$ in Lemma 6.3, and then describe how this can be used to obtain a realization for every multigraphical sequence fulfilling $\sum \mathbf{d} > 4(n-1)$ and $d_n \geq 2$.

▶ **Lemma 6.3** (⋆). *Let $\mathbf{d} = (d_1, \ldots, d_n)$ be a multigraphical sequence with $\sum \mathbf{d} = 4(n-1)$ and $d_n \geq 2$. We can compute a realization G of $\mathbf{d}$ and two disjoint spanning trees of G in $\mathcal{O}(n+m)$ time.*

Sketch. Intuitively, this result is again obtained via a constructive induction over the of n. Two cases are distinguished in the inductive step: If $d_n = 2$, then simply laying off d_n and reattaching it to the provided solution for the sequence of length $n-1$ provides a desired realization. By our data structure, this laying off and reattaching can be done by a constant time overhead. If $d_n = 3$, then the degree sequence actually is graphical and the algorithm behind Lemma 6.1 provides the desired result. ◀

The construction for sequences with a degree sum larger than $4(n-1)$ follows directly from the laying-off process for multigraphical sequences (Theorem 2.7), which reduces the degree sum by two while preserving the length of the sequence.

▶ **Lemma 6.4** (⋆). *Let $\mathbf{d} = (d_1, \ldots, d_n)$ be a multigraphical sequence with $\sum \mathbf{d} \geq 4(n-1)$ and $d_n \geq 2$. We can compute a realization G of $\mathbf{d}$ and two disjoint spanning trees of G in $\mathcal{O}(n+m)$ time.*

Realizations with spanning trees that share at most one edge and C_4-pivotable graph realizations. By carefully recalling the existential proofs of the previous sections, one can observe that these proofs imply linear-time constructions by using the data structure.

▶ **Lemma 6.5** (⋆). *Let $\mathbf{d}$ be a (multi)graphical degree sequence. If $\mathbf{d}$ is TC-realizable, then we can compute a realization of $\mathbf{d}$ in $\mathcal{O}(n+m)$ time.*

Computing a TC-labeling. We recall the labeling schemes for TC-labelings (that can be implemented in linear time) in the full version. We conclude the following algorithmic main result of our work.

▶ **Theorem 6.6** (⋆). *Let $\mathbf{d}$ be a (multi)graphical degree sequence. We can decide in $\mathcal{O}(n+m)$ time whether $\mathbf{d}$ is TC-realizable. If this is the case, then in $\mathcal{O}(n+m)$ time, we can compute a (multi)graph G which is a realization of $\mathbf{d}$ together with a TC-labeling for G.*

7 Conclusion

In this work, we presented complete characterizations of graphical and multigraphical degree sequences that admit temporally connected realizations under proper labelings. All characterizations presented in this paper are constructive and lead to linear-time algorithms for constructing TC realizations (including the corresponding labelings), provided that the input satisfies the necessary conditions.

As explained in introduction, the fact that we only considered proper labelings is not a limitation, as the remaining cases involving non-proper labelings are either trivial or covered by our techniques. Thus, these results cover directly or indirectly all the possible combinations of settings among proper/non-proper, simple/non-simple, and strict/non-strict.

Several natural questions could follow from the present work. For instance, one may ask to enumerate all TC-realizable graphs of a given degree sequence, or ask for realizations with additional constraints such as bounded lifetime, bounded diameter, or the existence of temporal spanners of a certain size. Another promising avenue is to optimize over these criteria while searching for a realization. Finally, the realizability question studied in this paper could be generalized to temporal k-connectivity or to directed temporal graphs.

References

1 Eleni C. Akrida, George B. Mertzios, Paul G. Spirakis, and Leszek Gasieniec. The Complexity of Optimal Design of Temporally Connected Graphs. *Theory of Computing Systems / Mathematical Systems Theory*, 61(3):907–944, April 2017. Publisher: Springer US. doi:10.1007/S00224-017-9757-X.

2 Davi de Andrade, Júlio Araújo, Allen Ibiapina, Andrea Marino, Jason Schoeters, and Ana Silva. Temporal Cycle Detection and Acyclic Temporization, March 2025. arXiv:2503.02694 [cs]. doi:10.48550/arXiv.2503.02694.

3 Brenda Baker and Robert Shostak. Gossips and Telephones. *Discrete Mathematics*, 2(3):191–193, June 1972. doi:10.1016/0012-365X(72)90001-5.

4 F. Boesch and F. Harary. Line removal algorithms for graphs and their degree lists. *IEEE Transactions on Circuits and Systems*, 23(12):778–782, December 1976. doi:10.1109/TCS.1976.1084170.

5 Filippo Brunelli, Pierluigi Crescenzi, and Laurent Viennot. Maximizing Reachability in a Temporal Graph Obtained by Assigning Starting Times to a Collection of Walks. *Networks*, 81(2):177–203, March 2023. doi:10.1002/net.22123.

6 Richard T Bumby. A problem with telephones. *SIAM Journal on Algebraic Discrete Methods*, 2(1):13–18, 1981.

7 Arnaud Casteigts, Timothée Corsini, and Writika Sarkar. Simple, Strict, Proper, Happy: A Study of Reachability in Temporal Graphs. *Theoretical Computer Science*, 991:114434, April 2024. doi:10.1016/j.tcs.2024.114434.

8 Michelle Döring. Simple, Strict, Proper, and Directed: Comparing Reachability in Directed and Undirected Temporal Graphs. In *Proceedings of the 36th International Symposium on Algorithms and Computation, ISAAC 2025*, Tainan, Taiwan, 2025. arXiv:2501.11697 [cs]. doi:10.48550/arXiv.2501.11697.

9 R. B. Eggleton and D. A. Holton. Simple and multigraphic realizations of degree sequences. In Kevin L. McAvaney, editor, *Combinatorial Mathematics VIII*, volume 884, pages 155–172. Springer Berlin Heidelberg, Berlin, Heidelberg, 1981. Series Title: Lecture Notes in Mathematics. doi:10.1007/BFb0091817.

10 Jessica Enright, Kitty Meeks, and Fiona Skerman. Assigning Times to Minimise Reachability in Temporal Graphs. *Journal of Computer and System Sciences*, 115:169–186, February 2021. doi:10.1016/j.jcss.2020.08.001.

11 Thomas Erlebach, Othon Michail, and Nils Morawietz. Recognizing and Realizing Temporal Reachability Graphs. In *Proceedings of the 33rd Annual European Symposium on Algorithms, ESA 2025*, LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.ESA.2025.93.

12 Thomas Erlebach, Nils Morawietz, and Petra Wolf. Parameterized Algorithms for Multi-Label Periodic Temporal Graph Realization. *Theoretical Computer Science*, page 115412, 2025. Publisher: Elsevier. doi:10.1016/j.tcs.2025.115412.

13 D. R. Fulkerson, A. J. Hoffman, and M. H. McAndrew. Some Properties of Graphs with Multiple Edges. *Canadian Journal of Mathematics*, 17:166–177, January 1965. doi:10.4153/CJM-1965-016-2.

14 Tibor Gallai and Paul Erdős. Gráfok előírt fokú pontokkal. *Matematikai Lapok*, 11:264–274, 1960.

15 Xiaofeng Gu, Hong-Jian Lai, and Yanting Liang. Multigraphic degree sequences and supereulerian graphs, disjoint spanning trees. *Applied Mathematics Letters*, 25(10):1426–1429, October 2012. doi:10.1016/j.aml.2011.12.016.

16 F. Göbel, J. Orestes Cerdeira, and H. J. Veldman. Label-Connected Graphs and the Gossip Problem. *Discrete Mathematics*, 87(1):29–40, January 1991. doi:10.1016/0012-365X(91)90068-D.

17 A. Hajnal, E. C. Milner, and E. Szemerédi. A Cure for the Telephone Disease. *Canadian Mathematical Bulletin*, 15(3):447–450, 1972. doi:10.4153/CMB-1972-081-0.

18 S. L. Hakimi. On Realizability of a Set of Integers as Degrees of the Vertices of a Linear Graph. I. *Journal of the Society for Industrial and Applied Mathematics*, 10(3):496–506, September 1962. Publisher: Society for Industrial and Applied Mathematics. doi:10.1137/0110037.

19 Frank Harary and Allen J. Schwenk. The Communication Problem on Graphs and Digraphs. *Journal of the Franklin Institute*, 297(6):491–495, January 1974. doi:10.1016/0016-0032(74)90126-4.

20 Václav Havel. Časopis pro pěstování matematiky. *Časopis pro pěstování matematiky*, 080(4):447–480, 1955. Publisher: Mathematical Institute of the Czechoslovak Academy of Sciences. URL: http://eudml.org/doc/19050.

21 Sandra M Hedetniemi, Stephen T Hedetniemi, and Arthur L Liestman. A survey of gossiping and broadcasting in communication networks. *Networks*, 18(4):319–349, 1988. doi:10.1002/NET.3230180406.

22 Juraj Hromkovic, Ralf Klasing, Burkhard Monien, and Regine Peine. Dissemination of information in interconnection networks (broadcasting and gossiping). *Combinatorial network theory*, 1:125–212, 1996.

23 D. J. Kleitman and D. L. Wang. Decomposition of a Graph Realizing a Degree Sequence into Disjoint Spanning Trees. *SIAM Journal on Applied Mathematics*, 30(2):206–221, 1976. Publisher: Society for Industrial and Applied Mathematics. URL: https://www.jstor.org/stable/2100523.

24 Nina Klobas, George B. Mertzios, Hendrik Molter, and Paul G. Spirakis. Temporal Graph Realization from Fastest Paths. *Theoretical Computer Science*, 1056:115508, 2025. doi:10.1016/j.tcs.2025.115508.

25 Sukhamay Kundu. Disjoint Representation of Tree Realizable Sequences. *SIAM Journal on Applied Mathematics*, 26(1):103–107, January 1974. doi:10.1137/0126006.

26 George B. Mertzios, Hendrik Molter, Nils Morawietz, and Paul G. Spirakis. Temporal graph realization with bounded stretch. In *Proceedings of the 50th International Symposium on Mathematical Foundations of Computer Science, MFCS 2025*, LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.MFCS.2025.75.

27 George B. Mertzios, Hendrik Molter, Nils Morawietz, and Paul G. Spirakis. Realizing temporal transportation trees. In *Proceedings of the 51st Workshop on Graph-Theoretic Concepts in Computer Science, WG 2025*, LNCS. Springer, 2025.

28 Julia Meusel, Matthias Müller-Hannemann, and Klaus Reinhardt. Directed Temporal Tree Realization for Periodic Public Transport: Easy and Hard Cases. In *Proceedings of the 25th Symposium on Algorithmic Approaches for Transportation Modeling, Optimization and Systems*, Warsaw, Poland, September 2025. OASIcs. arXiv:2504.07920 [cs]. doi:10.48550/arXiv.2504.07920.

29 Amitabha Tripathi, Sushmita Venugopalan, and Douglas B. West. A short constructive proof of the Erdős–Gallai characterization of graphic lists. *Discrete Mathematics*, 310(4):843–844, February 2010. doi:10.1016/j.disc.2009.09.023.

An Optimal Algorithm for the Stacker Crane Problem on Fixed Topologies

Yike Chen
University of Electronic Science and Technology of China, Chengdu, China

Ke Shi
University of Science and Technology of China, Hefei, China

Chao Xu
University of Electronic Science and Technology of China, Chengdu, China

Abstract

The Stacker Crane Problem (SCP) is a variant of the Traveling Salesman Problem. In SCP, pairs of pickup and delivery points are designated on a graph, and a crane must visit these points to move objects from each pickup location to its respective delivery point. The goal is to minimize the total distance traveled. SCP is known to be NP-hard, even on trees. The only positive results, in terms of polynomial-time solvability, apply to graphs that are topologically equivalent to a path or a cycle.

We propose an algorithm that is optimal for each fixed topology, running in near-linear time. This is achieved by demonstrating that the problem is fixed-parameter tractable (FPT) when parameterized by both the cycle rank and the number of branch vertices.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability; Theory of computation → Network flows; Mathematics of computing → Combinatorial optimization

Keywords and phrases Stacker Crane Problem, Fixed-Parameter Tractable, Min-Cost Circulation

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.18

Funding Supported by Science and Technology department of Sichuan Province under grant M112024ZYD0170.

Acknowledgements We would like to thank Siyue Liu and Jianbo Wang for the initial discussions related to this problem. Chao would like to thank Zexuan Liu and Luze Xu for their discussions on the proximity results of a related integer program.

1 Introduction

In the stacker crane problem (SCP), a stacker crane must retrieve and deliver a set of m items from and to specified locations within a warehouse. The goal is to determine an optimal order for these operations, constructing a tour that completes the task while minimizing the total distance traveled. The problem is typically modeled as a problem on a weighted graph, where the stacker crane moves between vertices, traversing the edges.

The SCP was first studied by Frederickson et al. [10], who provided an equivalent formulation involving a tour on a mixed graph and designed a 9/5-approximation algorithm. Subsequently, there have been many studies focused on developing approximation algorithms for various SCP variants [14, 18, 20].

Regarding exact algorithms, Frederickson and Guan demonstrated that SCP is NP-hard even on trees [8], ruling out some of the simplest graph classes as candidates for polynomial-time solvability. However, Atallah and Kosaraju showed that when the input graph is a path, SCP can be reduced to a minimum spanning tree problem. Moreover, if the input graph is a cycle with n vertices, there exists an $O(m + n \log n)$-time exact algorithm [2]. Frederickson later improved the running time to match that of finding a minimum spanning tree, and also proved the algorithm to be optimal [9].

© Yike Chen, Ke Shi, and Chao Xu;
licensed under Creative Commons License CC-BY 4.0

Since paths and cycles are topologically equivalent to a single edge or a self-loop, respectively, one might conjecture that SCP is solvable in polynomial time for any fixed topology. Fixed topologies have practical significance: the layout of warehouses is usually fixed, often resembling a grid with a limited number of aisles [4, 17]. For further details on various industrial applications of SCP, we refer readers to [6].

Our Contributions

We present an optimal algorithm for solving the SCP for fixed topologies. The idea is to design an FPT algorithm parameterized by the cycle rank and the number of branch vertices. Our algorithm generalizes Frederickson's approach, reducing to his algorithm when the topology is a path or a cycle [9]. Because our algorithm is more general, it also serves as an alternative and more intuitive proof of the correctness of Frederickson's algorithm for paths and cycles as special cases [9].

2 Preliminaries

We consider mixed graphs, which can contain both undirected edges and directed edges, the latter being referred to as *arcs*. A graph that contains only arcs is called a directed graph, while one that contains only undirected edges is called an undirected graph. A vertex is termed a *branch* vertex if its degree is at least 3.

For an arc (s, t), s is the tail and t is the head. For an undirected edge, both vertices are tails and heads. A *walk* is a sequence of edges such that the tail of the subsequent edge matches the head of the preceding edge. A *tour* is a walk that starts and ends at the same vertex. Let $\mathrm{MST}(m)$ denote the worst-case running time for finding a minimum spanning tree in a graph with m edges.

We now provide a graph-theoretical definition of SCP. In SCP, we are given a simple graph $B = (V, E)$, called the *base graph*, together with a list of *requests* R. A request is an ordered pair of vertices, represented as an arc. For simplicity, we assume that the requests are distinct, although our results can handle cases with duplicate requests.

We use the following formulation of SCP, as introduced by Frederickson et al. [10].

▶ **Problem 1** (Stacker Crane Problem (SCP)). *Given a simple graph $B = (V, E)$ on n vertices and a set of p arcs R, with a cost function $c : E \cup R \to \mathbb{R}^+$, find a min-cost tour in the mixed graph $G = (V, E \cup R)$ that traverses each arc in R exactly once.*

Intuitively, by assigning the cost of an arc $(s, t) \in R$ to be the length of the shortest st-path in B, traversing an arc represents moving an object from s to t. We emphasize that this ensures the object must be delivered immediately once it is picked up.

An *edge subdivision* operation involves inserting a new vertex in the middle of an edge. Specifically, given an edge uv, we delete the edge, add a new vertex w, and connect u and v via w by adding edges uw and wv. A graph H is called a *subdivision* of a graph G if H can be obtained through a sequence of edge subdivision operations starting from G. Two graphs, G_1 and G_2, are *topologically equivalent* if G_1 and G_2 are subdivisions of the same graph G. Our goal is to show that SCP for a fixed topology can be solved in optimal time.

▶ **Problem 2** (Stacker Crane Problem with Fixed Topology (SCP(H))). *Given a simple graph $B = (V, E)$ on n vertices that is topologically equivalent to a graph H, and a set of p arcs R, with a cost function $c : E \cup R \to \mathbb{R}^+$, find a min-cost tour in the mixed graph $G = (V, E \cup R)$ that traverses each arc in R exactly once.*

2.1 Circulations

Consider a mixed graph $G = (V, E \cup A)$ with a set of undirected edges E and directed arcs A. An arbitrary orientation is assigned to each undirected edge to define a forward direction, which serves solely as a notational convenience and has no bearing on the results. We define $\delta^-(v)$ and $\delta^+(v)$ as the sets of in-edges/in-arcs and out-edges/out-arcs of a vertex v, respectively. Each edge and arc has an associated lower bound, upper bound, and cost function, denoted as $\ell, u : E \cup A \to \mathbb{Z}$ and $c : E \cup A \to \mathbb{R}$.

A *circulation* is a function $f : E \cup A \to \mathbb{Z}$ such that $\sum_{e \in \delta^-(v)} f(e) = \sum_{e \in \delta^+(v)} f(e)$ for every vertex $v \in V$. The circulation f is *feasible* with respect to the lower bounds ℓ and upper bounds u if $\ell(e) \leq f(e) \leq u(e)$ for every edge $e \in E \cup A$. Typically, the values of ℓ and u are clear from the context.

The cost of a circulation f under the cost function c is given by $\sum_{a \in A} f(a)c(a) + \sum_{e \in E} |f(e)|c(e)$. A feasible circulation with the minimum cost is called a *min-cost circulation*. An edge or arc e is termed *fixed* if $\ell(e) = u(e)$. An edge e is considered *unconstrained* if $\ell(e) = -\infty$ and $u(e) = \infty$. An arc is *unconstrained* if $\ell(e) = 0$ and $u(e) = \infty$. The *support* of f, denoted by $\text{supp}(f)$, is defined as $\{e \mid f(e) \neq 0, e \in E \cup A\}$.

For a circulation f, we use $[f]$ to denote the set of all tours corresponding to f in a natural way: a tour where $f(e)$ represents the difference between the number of times the tour traverses the edge in the forward and backward directions. The set $[f]$ is referred to as a *homology class*, and two tours in $[f]$ are said to be *homologically equivalent*.

For the SCP problem with input graph $G = (V, E \cup R)$, the *corresponding circulation problem* is a circulation problem on G where each edge in E is unconstrained, and each arc in R is fixed with a flow value of 1. The costs of the edges and arcs remain the same as in the original SCP formulation. The SCP problem and the corresponding circulation problem are not equivalent. A gap exists due to connectivity. A tour must be connected, but the corresponding circulation does not guarantee connectivity by simply looking at the support.

2.2 Cycle space

For an undirected graph $G = (V, E)$, the functions $f : E \to \mathbb{Z}$ can be represented as vectors in $\mathbb{Z}^E$. The space of circulations is a submodule of $\mathbb{Z}^E$, known as the *integral cycle space*, or simply the cycle space. If the graph is connected, the dimension of the cycle space is $|E| - |V| + 1$ [3]. This value, $|E| - |V| + 1$, is referred to as the *cycle rank*. Any two topologically equivalent graphs have the same cycle rank.

A circulation f is called a *cycle flow* if $\text{supp}(f)$ forms an undirected cycle (after considering all edges as undirected). A cycle flow f is a *unit cycle flow* if $|f(e)| = 1$ for all $e \in \text{supp}(f)$. Let T be a spanning tree in G. For each edge $uv \notin T$, the *fundamental cycle* C_{uv} with respect to T is defined as the unit cycle flow that traverses uv in the forward direction and then traces back from v to u along the unique path in T. Note that it is possible that some fundamental cycles have a negative flow value on some edges.

The fundamental cycles with respect to T form a basis for the cycle space. That is, let $C_1, \ldots, C_r$ be the fundamental cycles with respect to T. Then, for any circulation f, we have $f = \sum_{i=1}^{r} \lambda_i C_i$ for some coefficients $\lambda \in \mathbb{Z}^r$ [15].

2.3 Techniques

Atallah and Kosaraju solved SCP(H) for the cases where H is either a path or a cycle [2]. Using a more modern and general perspective, their approach can be summarized as a three-step algorithm:

1. Find a min-cost circulation f in the corresponding circulation problem.
2. Find all circulations g such that $\|f - g\|_\infty \leq b$ for some bound b.
3. Find a min-cost tour in each homology class $[g]$, and return the minimum.

It is crucial that H is a path or a cycle, as their technique in each of these steps relies heavily on this property. We describe how to eliminate such a restriction.

In the first step, finding a min-cost circulation when H is either a path or a cycle was solved through an ad-hoc process in *linear time*. On a path, such a circulation is unique. For a cycle, they fix a flow sent along a particular edge, and then the circulation becomes unique. Alternatively, one can use a min-cost flow algorithm on a one-tree [13], which can also obtain a linear time algorithm. We show that min-cost circulation can be solved in linear time for arbitrary fixed H by reducing it to a problem similar to linear programming in a fixed dimension. See Subsection 4.1.

For the second step, Atallah and Kosaraju showed that $b = 0$ suffices for paths, leading to an optimal algorithm. In the same paper, they established a bound of $b = p$ for cycles, where p is the number of requests. Fortunately, their technique does not explicitly generate all circulations, but instead searches through them using a binary search, implemented alongside step 3. Later, Frederickson improved the bound on b to 1 by squeezing the value of the min-cost tour between two one-dimensional functions, thereby achieving the desired proximity result [9]. This improvement not only made the algorithm faster but also allowed steps 2 and 3 to be separated.

In general, it is not difficult to argue that for any H, a bound of $b = p$ suffices. The number of feasible circulations in step 2 would then be $O((2p+1)^r)$, where r is the cycle rank of B. This is too large to yield a fast algorithm. Unfortunately, Frederickson's technique does not generalize to graphs with higher cycle ranks. Therefore, we need a proximity theorem for arbitrary graphs. This is the most technical part of our paper. As we will show in Section 3, a bound of $b = r$ suffices. The algorithm to explicitly enumerate $O((2r+1)^r)$ such circulations will be described in Subsection 4.2.

For the third step, Atallah and Kosaraju found the min-cost tour in a homology class by computing a minimum spanning tree in an auxiliary graph. The auxiliary graph is constructed by contracting edges in the circulation that defines the homology class. Once again, computing a minimum spanning tree suffices only because H is a path or a cycle. In Subsection 4.3, we show that the correct problem to solve is *minimum Steiner tree* in the auxiliary graph, which is an NP-hard problem in general. However, since the number of Steiner branch vertices depends only on H, the running time remains efficient.

3 Proximity result

Let $B = (V, E)$ be the base graph. Consider the mixed graph $G = (V, E \cup R)$ in the SCP problem and the corresponding circulation problem on G. Assume f is the min-cost circulation in G. We will show that there exists a circulation g in G such that the homology class $[g]$ contains the min-cost tour, and $\|f - g\|_\infty \leq r$, where r is the cycle rank of B.

For a unit cycle flow C, we call kC a cycle flow of value k. A circulation f is said to contain a circulation g if $|g(e)| \leq |f(e)|$ for each edge e, and $g(e)$ and $f(e)$ have the *same sign*.

▶ **Definition 1.** *A circulation is* elementary *if it does not contain a cycle flow of value* 2.

Intuitively, this means that after cancellations, no part of the flow circulates around a cycle more than once. In fact, we prove the proximity result by showing that $f - g$ is an elementary circulation in B, the undirected base graph.

3.1 Elementary Circulations

For a graph or mixed graph G, let $G' = \text{sym}(G)$ be the directed graph obtained by replacing each edge with two opposing arcs. Interestingly, we prove the proximity result by considering circulations in the residual graph of G', as it is easier to reason about.

The *residual graph* for a given directed graph G with lower and upper bounds ℓ and u, and a feasible circulation f, is the directed graph G_f along with new lower and upper bounds ℓ_f and u_f. For each arc $a \in A$, define $u_f(a) = u(a) - f(a)$ and $\ell_f(a) = \ell(a) - f(a)$. The residual graph G_f consists of all arcs for which either $u_f(a) \neq 0$ or $\ell_f(a) \neq 0$. Note that the lower bounds in the residual graph do not have to be non-negative. However, $\ell_f(a) \leq 0 \leq u_f(a)$ is always satisfied, meaning that the zero circulation is always feasible. The fixed arcs in the original graph do not appear in the residual graph, so the complexity of the residual graph can be much smaller than the original graph.

One can observe that the SCP can be reduced to a min-cost *connected* circulation problem on $G' = (V, A \cup R) = \text{sym}(G)$. Here, A are the arcs obtained from symmetrizing E. For arcs in A, the lower bound is 0 and the upper bound is infinity. The cost of each arc is the cost of the corresponding edge. For arcs in R, both the lower bound and upper bound are 1. If f' is a min-cost connected circulation in G', then it corresponds to a min-cost tour of the same cost in G.

▶ **Theorem 2.** *Let f be a min-cost circulation in $G = (V, E \cup R)$, where the flows on arcs in R are fixed to 1, and edges E are unconstrained. There exists a circulation g such that the min-cost tour is in the homology class $[g]$ and $g - f$ is an elementary circulation in the base graph $B = (V, E)$.*

Proof. Let $\Delta = g - f$. We choose g such that a min-cost tour is contained in $[g]$, and out of all such g, one that minimizes $\|\Delta\|_1$. Note that Δ is a circulation in B because f and g agree on the fixed arcs in R, hence Δ is zero outside of E.

Let $G' = \text{sym}(G)$, and let f' be the min-cost circulation in G', which corresponds to the min-cost circulation f in G in a natural way. Specifically, for each edge uv, let (u, v) denote the positive orientation. Define $f'(u, v) = f(uv)$ and $f'(v, u) = 0$ if $f(uv) \geq 0$ and otherwise set $f'(u, v) = 0$ and $f'(v, u) = -f(uv)$. This ensures that $f(uv) = f'(u, v) - f'(v, u)$. Let g' be the min-cost connected circulation in G', such that $g(uv) = g'(u, v) - g'(v, u)$, again using the positive orientation (u, v). Let $\Delta' = g' - f'$, which is a circulation in $B' = \text{sym}(B)$.

Consider the residual graph $G'_{f'}$. Certainly, Δ' is a feasible circulation in $G'_{f'}$ because $f' + \Delta' = g'$ is a feasible circulation in G'. We label the arcs in $G'_{f'}$ as follows: for each edge $e \in E$, label two corresponding arcs as e^+ and e^- such that $\Delta'(e^+) - \Delta'(e^-) \geq 0$, which implies $\Delta'(e^+) - \Delta'(e^-) = |g'(e^+) - f(e^+) - g'(e^-) + f(e^-)| = |g(e) - f(e)| = |\Delta(e)|$.

Assume Δ is not elementary. Then, there exists a unit cycle flow C such that $2C$ is contained in Δ. In terms of $G'_{f'}$, this would imply that $\Delta'(e^+) - \Delta'(e^-) \geq 2$ for each $e \in C$. We consider a cycle flow C' in $G'_{f'}$ whose undirected version is C, and which uses the maximum number of edges with positive labels. That is, for each edge e, we set $C'(e^+) = 1$ and $C'(e^-) = 0$ if $\Delta'(e^+) \geq 2$, otherwise, since $\Delta'(e^+) \leq 1$, it follows that $\Delta'(e^-) \leq \Delta'(e^+) - 2 \leq -1$, set $C'(e^+) = 0$ and $C'(e^-) = -1$.

Certainly, $\Delta' - C'$ is feasible in $G'_{f'}$. The cost of C' cannot be negative, since the residual graph of a min-cost circulation cannot contain a negative cost cycle [1]. This implies that the cost of $g' - C'$ is no greater than the cost of g'.

Additionally, we need to show that $\text{supp}(g' - C')$ is connected. For each $e \in \text{supp}(C)$, there are two cases:

1. If $C'(e^+) = 1$, then $\Delta'(e^+) \geq 2$. Therefore, $g'(e^+) - C'(e^+) = f'(e^+) + \Delta'(e^+) - 1 \geq 1$. Hence, $e^+ \in \text{supp}(g' - C')$.
2. If $C'(e^-) = -1$, then $\Delta'(e^-) \leq -1$. Therefore, $g'(e^-) - C'(e^-) = g'(e^-) + 1 \geq 1$. Hence, $e^- \in \text{supp}(g' - C')$.

Since $\text{supp}(g' - C')$ would include at least one of e^+ or e^- for each $e \in C$, if $\text{supp}(g')$ is connected, then $\text{supp}(g' - C')$ is also connected. Because $g' - C'$ is a connected circulation of no greater cost than g', we have that $[g - C]$ also contains a min-cost tour. However, $\|\Delta - C\|_1 < \|\Delta\|_1$ because Δ contains C. This contradicts our choice of Δ having minimal L_1 norm. ◀

3.2 Elementary Circulations and L_∞ Norm

In this section, we bound the number of elementary circulations in a *graph* using its cycle rank.

The idea is to show that for an elementary circulation f, we have $\|f\|_\infty \leq r$, where r is the cycle rank, by expressing the circulation as the sum of r cycle flows. We then use the fact that no flow traverses a cycle more than once to establish the desired bound.

However, a naive decomposition might result in a cycle flow with values greater than 2, even though the overall circulation remains elementary due to cancellations.

To avoid this issue, we transition to non-negative circulations, ensuring that everything we analyze is non-negative. This simplifies the argument, as it prevents flow cancellations on edges when summing flows around cycles.

First, we strengthen the well-known flow decomposition theorem for circulations, which typically states that a circulation can be decomposed into m cycle flows, where m is the number of edges [19]. We show that this can be improved to r, the cycle rank of the graph.

▶ **Proposition 3.** *Every non-negative circulation on a graph with cycle rank r can be decomposed into at most r non-negative cycle flows. That is, $f = \sum_{i=1}^{r} \lambda_i C_i$, where $C_1, \ldots, C_r$ are non-negative cycle flows and $\lambda_1, \ldots, \lambda_r \geq 0$.*

Proof. We prove this by induction on the cycle rank r.

If the cycle rank is 0, then the graph contains no cycles, and f must be zero everywhere.

Now, consider the case where the cycle rank is r. We can assume the graph is connected; otherwise, we can apply the proof to each connected component separately. Suppose there exists a cycle C with positive flow. Let $\lambda = \min_{e \in C} f(e)$. We include λC as a term in the decomposition.

Let D be the set of edges with zero flow after reducing the flow on C by λ. Assume that $G - D$ has k components. Observe that $k \leq |D| + 1$. If $k = |D| + 1$, this would imply that removing each edge in D disconnects the graph, contradicting the fact that G contains the cycle C. Hence, $k \leq |D|$.

Let the components of $G - D$ be $V_1, \ldots, V_k$. By the inductive hypothesis, the remaining circulation can be decomposed into $\sum_{i=1}^{k} (|E_i| - |V_i| + 1) = (m - |D|) - n + k \leq m - n = r - 1$ non-negative cycle flows. Therefore, we can decompose f into r non-negative cycle flows. ◀

Next, we show that elementary circulations have a small L_∞ norm.

▶ **Proposition 4.** *Let f be an elementary circulation in a graph with cycle rank r. Then, $\|f\|_\infty \leq r$.*

Proof. Let f be an elementary circulation. Assume f is non-negative; if not, we reverse the orientation of the edges with negative flow so that f becomes non-negative. By Proposition 3, f can be decomposed into r non-negative cycle flows. Since f is an elementary circulation, each of the r cycle flows is a unit cycle flow. Hence, for each edge e, we have $f(e) = \sum_{i=1}^{r} C_i(e) \leq \sum_{i=1}^{r} 1 = r$. Therefore, $\|f\|_\infty \leq r$. ◀

The bound in Proposition 4 is tight. Consider a cycle rank r graph with $2(r+1)$ vertices that consists of a path $P = v_1, \ldots, v_{2r}$. There are also edges e_{r-i} between v_{2r-i} and v_{1+i} for $i \leq r$. See Figure 1. It has cycle rank r. Let C_i be the fundamental cycle with respect to the path P using edge e_i in the smaller to larger vertex direction, where $i \geq 1$. Consider the circulation $f = \sum_{i=1}^{r} C_i$, which is elementary, and $f(e_0) = r$.

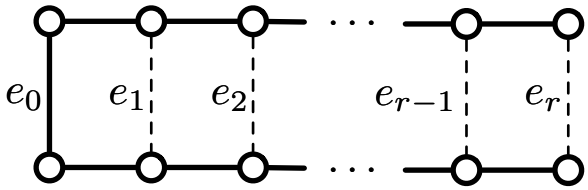


Figure 1 Example graph where there is an elementary circulation where $\|f\|_\infty = r$.

4 The new algorithm

This section describes the optimal algorithm up to constants for SCP(H).

4.1 Min-Cost Circulation

Recall that we are interested in finding the min-cost circulation f of $G = (V, E \cup R)$, where $f(a) = 1$ for all $a \in R$. The min-cost circulation problem was shown to be solvable in $m^{1+o(1)}$ time, where m is the number of edges in the graph [5]. In our case, the graph has $p + m$ edges. The min-cost flow algorithm on a one-tree gives a linear time algorithm for the special case when the base graph have cycle rank 1 [13]. We can show that there is a linear time algorithm for base graphs with fixed cycle rank.

▶ **Theorem 5.** *Let r be a fixed number. Consider a min-cost circulation problem where the edges are either fixed or unconstrained, and the unconstrained edges form a graph with cycle rank r. The min-cost circulation can be found in $O(m)$ time.*

Proof. Let $B = (V, E)$ be the undirected graph containing all the unconstrained edges, and let R be the set of fixed edges. We have $G = (V, E \cup R)$. Consider a spanning tree T on B, and let the edges in $E \setminus T$ be $e_1, \ldots, e_r$. We define $g(\lambda_1, \ldots, \lambda_r)$ to be the cost of the min-cost circulation f in G such that $f(e) = 1$ for $e \in R$ and $f(e_i) = \lambda_i$ for $1 \leq i \leq r$. To find the min-cost circulation, we need to solve $\min_\lambda g(\lambda)$. Note that there is an integral minimizer, as λ_i represents the flow value on edge e_i, which can always be taken to be integral.

Finding the minimum is equivalent to solving the following linear program. For each $a \in R$, compute the fundamental cycle C_a, and let $b_e = \sum_{e \in C_a} C_a(e)$. For each $e \in E$ and $1 \leq i \leq r$, we define $A_{e,i} = C_{e_i}(e)$.

Here, A is the edge-fundamental cycle incidence matrix, which is a network matrix [15].

$$\begin{aligned} &\min_{\lambda \in \mathbb{R}^{E \setminus T}, x \in \mathbb{R}^T} && \sum_e x_e \\ &\text{subject to} && x_e \geq c(e) \left| A_e \cdot \lambda + b_e \right| \quad \forall e \in E \end{aligned}$$

Zemel showed that the minimizer of mathematical programs of the above form can be seen as a generalization of r-dimensional L_1 linear regression, which can be solved using techniques similar to Megiddo's constant-dimension linear programming algorithm [16]. Consequently, the problem can be solved in $2^{O(2^r)}m = O(m)$ time [21]. ◀

4.2 Generating Circulations Near the Min-Cost Circulation

In this section, we show how to generate all circulations that are close to the min-cost circulation in terms of the L_∞ norm.

▶ **Lemma 6.** *For an undirected graph with cycle rank r, the number of circulations f such that $\|f\|_\infty \leq k$ is at most $(2k+1)^r$ and these circulations can be found in $O((2k+1)^r m)$ time.*

Proof. Consider any fundamental cycle basis of G, a graph with cycle rank r, with respect to some spanning tree T. The values on the edges outside T uniquely determine the circulation. For a circulation f such that $\|f\|_\infty \leq k$, the absolute value of the flow on each edge outside T is at most k. Hence, there can be at most $2k+1$ choices for each of the r edges outside T. This shows that there can be at most $(2k+1)^r$ circulations with L_∞ norm at most k.

To construct all such circulations, we find a fundamental cycle basis $C_1, \ldots, C_r$. This takes $O(rm)$ time. Next, we generate circulations of the form $\sum_{i=1}^r \lambda_i C_i$ one by one where $-k \leq \lambda_i \leq k$. The sequence of circulations is generated using a generalized Gray code, so two consecutive circulations differ in only a single cycle [11]. Therefore, it takes $O(m)$ time to generate a circulation from the previous circulation by augmenting a cycle. The total running time for generating all circulations with L_∞ norm no larger than k is $O((2k+1)^r m)$. ◀

▶ **Corollary 7.** *Let f be the min-cost circulation in G. The number of circulations g in G such that $\|f - g\|_\infty \leq r$ is $O((2r+1)^r)$ and they can be found in $O((2r+1)^r m)$ time, where m is the number of edges in G.*

Proof. By Lemma 6, there can be at most $(2r+1)^r$ circulations with L_∞ norm no larger than r in B. These circulations can be computed in $O((2r+1)^r m)$ time. For each such circulation h, we compute $g = f + h$. This takes an additional $O(m)$ time per circulation. Thus, the total time complexity is $O((2r+1)^r m)$. ◀

4.3 Min-Cost Tour in a Given Homology Class

Recall that we are interested in finding the min-cost tour in the homology class $[f]$ for some given circulation f in $G = (V, E \cup R)$, where $f(a) = 1$ for $a \in R$.

If f is already connected, then we are done, as we can find a tour that uses each edge e exactly $|f(e)|$ times. Otherwise, we need to find a tour in $[f]$. Note that in addition to the edges in $\mathrm{supp}(f)$, some extra edges might need to be used by the tour. Naturally, this would look like a Steiner Asymmetric Traveling Salesman Problem (ATSP), defined as finding the shortest tour in a directed graph that visits each terminal vertex at least once, which is even harder to work with than ATSP. However, because we are restricting the tour to be in $[f]$, this limits the kinds of edges we add to the tour: Specifically, these are edges outside $\mathrm{supp}(f)$ that must be traversed both forward and backward *exactly once*.

To address this, we take G and contract each connected component of $\mathrm{supp}(f)$ into a single vertex. Let the resulting graph be B'. Note that only edges in E might remain, as all arcs in R have been contracted. Therefore, B' is a graph instead of a mixed graph. Moreover, we change the weight of the edges to twice their original cost since each edge would be traversed once forward and once backward. We then reduce the problem to the minimum Steiner tree problem.

For a graph $G = (V, E)$ with a set of terminal vertices $T \subseteq V$, the vertices in $V \setminus T$ are called *Steiner vertices*. A Steiner tree is a tree that contains all the vertices in T. The minimum Steiner tree with respect to a weight function $w : E \to \mathbb{R}^+$ is a Steiner tree of minimum weight.

We are interested in solving the minimum Steiner tree problem on B', where the terminals are the contracted vertices. This would give us the minimum set of extra edges the tour must traverse to be connected. The minimum Steiner tree problem is NP-hard in general, but note that B' has only a bounded number of Steiner branch vertices. The number of Steiner branch vertices in B' is at most the number of branch vertices in B. Indeed, the contracted vertices are not Steiner vertices, so all Steiner branch vertices must be branch vertices in B. The Steiner vertices of degree 1 and 2 can be preprocessed away, so only Steiner branch vertices remain [7]. Hence, the minimum Steiner tree can be solved in $O(2^k \,\mathrm{MST}(m))$ time for a graph containing k Steiner branch vertices and m edges [12].

Since B' can have at most m edges, the minimum Steiner tree in B' can be found in $O(2^k \,\mathrm{MST}(m))$ time. This leads us to the following theorem.

▶ **Theorem 8.** *For an input mixed graph $G = (V, E \cup R)$, let f be a circulation where $f(a) = 1$ for each $a \in R$. Finding a min-cost tour in $[f]$ takes $O(p + 2^k \,\mathrm{MST}(m))$ time, where $m = |E|$, $p = |R|$, and k is the number of branch vertices in $B = (V, E)$.*

4.4 Putting Everything Together

Combining all the components, we obtain the desired algorithm as described in Figure 2. We are now ready to prove our main theorem.

▶ **Theorem 9.** *SCP(H) with an input graph of n vertices and p requests can be solved in $O(\mathrm{MST}(n) + p)$ time.*

Proof. Let B be the base graph and R be the set of requests. For a fixed topology, the cycle rank r and the number of branch vertices k are constants. If m is the number of edges in B, then $m = n + r - 1 = O(n)$.

Consider the algorithm in Figure 2. By Theorem 5, computing the min-cost circulation takes $O(m + p)$ time. By Corollary 7, enumerating all $O(1)$ possible circulations close to the min-cost circulation takes $O(m)$ time. By Theorem 8, finding the min-cost tour in the homology class of each circulation takes $O(\mathrm{MST}(m) + p)$ time. Using the fact that $m = O(n)$, we obtain a final running time of $O(\mathrm{MST}(n) + p)$. ◀

The running time in Theorem 9 is optimal. The term $\mathrm{MST}(n)$ is unavoidable, as finding the minimum spanning tree on n edges can be reduced to $\mathrm{SCP}(P)$ in linear time, where P is a path [9]. Moreover, since reading the input requires at least $O(p)$ time, the term p is also tight.

Now, in terms of FPT, what we showed is that the SCP problem with the cycle rank and the number of branch vertices as parameters is FPT.

```
SCP(B = (V, E), R, c):
  G ← (V, E ∪ R)
  f ← the min-cost circulation
  C ← all circulations g such that ‖f − g‖∞ ≤ r
  for g ∈ C
      record MinCostTourInHomologyClass(G, g, c) as a candidate
  return the min-cost candidate
```

Figure 2 Pseudocode for solving SCP given base graph B, requests R and cost function c.

```
MinCostTourInHomologyClass(G, f, c):
  ⟨⟨Find the min-cost tour in [f]⟩⟩
  H ← contract all components of supp(f) in G
  for each edge e ∈ V(H)
      w(e) ← 2c(e)
  T ← the vertices from contracting a component of supp(f)
  S ← Minimum Steiner tree on H with weights w and terminals T
  W ← construct a tour from S and f
  return W
```

Figure 3 Finding the min-cost tour for a given graph G and a circulation f on G.

5 Discussion

We simplified the presentation of our results, but our approach can be extended in multiple ways without increasing the time complexity.

As mentioned earlier in the preliminaries, we can handle duplicate arcs in R without altering the complexity of the algorithm. Indeed, this is done by allowing demands on arcs in R. Specifically, we can introduce a function $d : R \to \mathbb{N}$ such that, for each $a \in R$, the tour is required to traverse a exactly $d(a)$ times. This generalization can be handled by simply fixing the value of the flow to be $d(a)$ for each $a \in R$ during our computation.

Our current cost function is symmetric; for an edge $e = uv$, traversing from u to v or from v to u incurs the same cost. However, our algorithm can be adapted to work with asymmetric cost functions as well by splitting the edge into two edges of opposite orientation with different costs and only allowing non-negative flows. During Steiner tree computations, merge the two edges and give the merged edge a weight equal to the cost of traversing the edge forward and backward once.

Although we provide an optimal algorithm, the constant factor hidden in the time complexity is extremely large, making it impractical for real-world applications. If the fixed topology has cycle rank r and k branch vertices, then the hidden factor is $2^{O(2^r)} + (2r+1)^r \cdot 2^k$. The bottleneck of $2^{O(2^r)}$ arises from solving the min-cost flow problem on a graph that is a tree with r additional edges. In practice, one can use theoretically slower, off-the-shelf min-cost flow algorithms to avoid the exponential dependency on r. However, there are two potential avenues for improvement:

1. The problem is very close to linear programming in constant dimensions, so techniques other than Megiddo's algorithm might be beneficial for improving the running time.
2. In the proof of Theorem 5, the matrix A is a *network matrix*, but Zemel's algorithm is designed for arbitrary matrices. This presents another opportunity for optimization.

The factor $(2r+1)^r$ is unlikely to be significantly improved if we need to enumerate all elementary circulations, as their number can be exponential with respect to r. However, there may be strategies to consider only a much smaller subset of elementary circulations. For example, our enumeration of elementary circulations is independent of cost information, which could potentially be leveraged to reduce the search space.

References

1 Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. *Network flows*. Pearson, Upper Saddle River, NJ, February 1993.

2 Mikhail J. Atallah and S. Rao Kosaraju. Efficient Solutions to Some Transportation Problems with Applications to Minimizing Robot Arm Travel. *SIAM Journal on Computing*, 17(5):849–869, October 1988. doi:10.1137/0217053.

3 Bela Bollobas. *Modern graph theory*. Graduate Texts in Mathematics. Springer, New York, NY, 1998 edition, December 2013.

4 Hadrien Cambazard and Nicolas Catusse. Fixed-parameter algorithms for rectilinear steiner tree and rectilinear traveling salesman problem in the plane. *European Journal of Operational Research*, 270(2):419–429, 2018. doi:10.1016/j.ejor.2018.03.042.

5 Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 612–623, 2022. doi:10.1109/FOCS54457.2022.00064.

6 Ángel Corberán and Gilbert Laporte. *Arc Routing*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2015. doi:10.1137/1.9781611973679.

7 Cees Duin. *Preprocessing the Steiner Problem in Graphs*, pages 175–233. Springer US, Boston, MA, 2000. doi:10.1007/978-1-4757-3171-2_10.

8 G.N. Frederickson and D.J. Guan. Nonpreemptive Ensemble Motion Planning on a Tree. *Journal of Algorithms*, 15(1):29–60, July 1993. doi:10.1006/jagm.1993.1029.

9 Greg N. Frederickson. A Note on the Complexity of a Simple Transportation Problem. *SIAM Journal on Computing*, 22(1):57–61, February 1993. doi:10.1137/0222005.

10 Greg N. Frederickson, Matthew S. Hecht, and Chul E. Kim. Approximation algorithms for some routing problems. *SIAM Journal on Computing*, 7(2):178–193, 1978. doi:10.1137/0207017.

11 Dah-Jyh Guan. Generalized gray codes with applications. In *Proc. Natl. Sci. Counc. Repub. China Part A Phys. Sci. Eng.*, volume 22(6), pages 841–848. Citeseer, 1998.

12 S. L. Hakimi. Steiner's problem in graphs and its implications. *Networks*, 1(2):113–133, 1971. doi:10.1002/net.3230010203.

13 Bahman Kalantari and Iraj Kalantari. *A Linear-Time Algorithm for Minimum Cost Flow on Undirected One-Trees*, pages 217–223. Springer US, Boston, MA, 1995. doi:10.1007/978-1-4613-3554-2_13.

14 Jianping Li, Xiaofei Liu, Weidong Li, Li Guan, and Junran Lichen. Approximation algorithms for the generalized stacker crane problem. In Xiaofeng Gao, Hongwei Du, and Meng Han, editors, *Combinatorial Optimization and Applications*, pages 95–102, Cham, 2017. Springer International Publishing. doi:10.1007/978-3-319-71150-8_8.

15 Christian Liebchen and Romeo Rizzi. Classes of cycle bases. *Discrete Applied Mathematics*, 155(3):337–355, 2007. doi:10.1016/j.dam.2006.06.007.

16 Nimrod Megiddo. Linear programming in linear time when the dimension is fixed. *J. ACM*, 31(1):114–127, January 1984. doi:10.1145/2422.322418.

17 Kees Jan Roodbergen and René de Koster. Routing order pickers in a warehouse with a middle aisle. *European Journal of Operational Research*, 133(1):32–43, 2001. doi:10.1016/S0377-2217(00)00177-6.

18 Yuhui Sun, Wei Yu, and Zhaohui Liu. Approximation algorithms for some min–max and minimum stacker crane cover problems. *Journal of Combinatorial Optimization*, 45(1):18, 2022. doi:10.1007/s10878-022-00955-x.

19 David P. Williamson. *Network Flow Algorithms*. Cambridge University Press, 2019.

20 Wei Yu, Rui-Yong Dai, and Zhao-Hui Liu. Approximation Algorithms for Multi-vehicle Stacker Crane Problems. *Journal of the Operations Research Society of China*, 11(1):109–132, March 2023. doi:10.1007/s40305-021-00372-7.

21 Eitan Zemel. An O(n) algorithm for the linear multiple choice knapsack problem and related problems. *Information Processing Letters*, 18(3):123–128, March 1984. doi:10.1016/0020-0190(84)90014-0.

Traffic-Oblivious Multi-Commodity Flow Network Design

Markus Chimani
Theoretical Computer Science, Osnabrück University, Germany

Max Ilsen[1]
Theoretical Computer Science, Osnabrück University, Germany

Abstract

We consider the Minimum Multi-Commodity Flow Subgraph (MMCFS) problem: given a directed graph G with edge capacities *cap* and a retention ratio $\alpha \in (0, 1)$, find an edge-wise minimum subgraph $G' \subseteq G$ such that for all traffic matrices T routable in G using a multi-commodity flow, $\alpha \cdot T$ is routable in G'. This natural yet novel problem is motivated by recent research that investigates how the power consumption in backbone computer networks can be reduced by turning off connections during times of low demand without compromising the quality of service. Since the actual traffic demands are generally not known beforehand, our approach must be traffic-oblivious, i.e., work for all possible sets of simultaneously routable traffic demands in the original network.

In this paper we present the problem, relate it to other known problems in literature, and show several structural results, including a reformulation, maximum possible deviations from the optimum, and NP-hardness (as well as a certain inapproximability) already on very restricted instances. The most significant contribution is a $\max(1/\alpha, 2)$-approximation based on a surprisingly simple LP-rounding scheme. We also give instances where this worst-case approximation ratio is met and thus prove that our analysis is tight.

2012 ACM Subject Classification Mathematics of computing → Network flows; Mathematics of computing → Approximation algorithms; Networks → Network design and planning algorithms

Keywords and phrases Multi-commodity flow, Digraphs, LP-rounding, Approximation algorithm

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.19

Related Version *Previous Version*: https://arxiv.org/abs/2504.16744v1 [16]

Funding Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) grant 461207633 (CH 897/7-2).

1 Introduction

We present the (suprisingly seemingly novel) network design problem Minimum Multi-Commodity Flow Subgraph (MMCFS): given a directed flow network G with edge capacities and a *retention ratio* $\alpha \in (0, 1)$, find a subnetwork $G' \subseteq G$ of minimum size such that G' still allows for a multi-commodity flow (MCF) routing of *any* traffic demands routable in G when they are scaled down by factor α.

The problem arises naturally in recent research concerning power saving in backbone (Tier 1) networks of Internet service providers. There, the overall amount of traffic has distinct peaks in the evenings (when people are, e.g., streaming videos) and lows late at night and in the early mornings [40]. Clearly, the networks are built to handle the peak times. This opens up the possibility to reduce the power consumption of the network by turning off some resources – e.g., connections, line cards, or servers – during low traffic periods [44, 14].

[1] Corresponding author.

© Markus Chimani and Max Ilsen;
licensed under Creative Commons License CC-BY 4.0

So consider a computer network that allows for the simultaneous routing of the traffic at peak times. This traffic is comprised of a set of *commodities*, where each commodity is identified by a pair of nodes (s, t) in the network and has a *demand* d specifying that d flow units have to be sent from s to t. The entirety of the demands for each pair of nodes is encoded in a *traffic matrix* T. Commonly, one makes the simplifying assumption that during low traffic periods the traffic demands are upper bounded by a down-scaling of T using a factor $\alpha \in (0, 1)$; the most practically relevant scenarios concern $\alpha \geq \frac{1}{2}$ [34]. The task now is to minimize the size of the network by deactivating connections such that the reduced network still accommodates a routing of the scaled-down demands $\alpha \cdot T$. However, in practice the traffic matrix T may change from day to day (in fact, even within sampling windows of 15 minutes) and while we can assume the capacity of the network to be large enough for all occurring traffic at any given time, T is usually not known beforehand. Thus, our solution should be *traffic-oblivious*, i.e., independent of any specific traffic matrix.

Technically, routing in realistic scenarios is *not* done via fully general multi-commodity flows: while MCF would be optimal in terms of minimizing congestion [40], it would be too complicated and temporally unstable for the routing hardware. Instead, simpler techniques like *2-segment routing* are used (which, in contrast to trivial shortest path routing, routes flow along a sequence of two chained shortest subpaths) [19]. Interestingly, studies show that in realistic networks, these routing solutions are virtually identical to those achieved by MCF [40, 12]. At the same time, for a given fixed network and traffic matrix, an MCF can be computed in polynomial time while establishing an optimal routing table for 2-segment routing (which is then deployable on router hardware) is NP-hard [25]. Thus, we describe the feasibility of solutions in our problem setting in terms of routability via MCF, and assume that realistic (simpler) routing protocols will still be able to attain effective routability.

Let us give a formal description of our problem: Given a directed graph (or *digraph*) $G = (V, E)$ with positive edge capacities $cap\colon E \to \mathbb{Q}$ and a traffic matrix T, a *flow* $f_{s,t}\colon E \to \mathbb{Q}$ from a vertex $s \in V$ to a vertex $t \in V$ is a function satisfying the flow conservation constraints

$$\sum_{uv \in E} f_{s,t}(uv) - \sum_{vu \in E} f_{s,t}(vu) = \begin{cases} -T(s,t) & \text{if } v = s \\ T(s,t) & \text{if } v = t \\ 0 & \text{else} \end{cases} \qquad \forall v \in V.$$

A *multi-commodity flow (MCF)* is a set of flows $\mathcal{F} = \{f_{s,t} \mid (s,t) \in V^2\}$ satisfying

$$\sum_{(s,t) \in V^2} f_{s,t}(uv) \leq cap(uv) \qquad \forall uv \in E.$$

For an edge $e = st \in E$, we may use the shorthand notation $T(e) \coloneqq T(s,t)$. Further, we call a traffic matrix T *routable in an edge set* $A \subseteq E$ if there exists an MCF $\mathcal{F}$ for (G, cap, T) with $\sum_{f \in \mathcal{F}} f(e) = 0$ for all $e \notin A$. Based on this notion, we define the Minimum Multi-Commodity Flow Subgraph (MMCFS) problem as follows: given a digraph $G = (V, E)$ with edge capacities cap and a retention ratio $\alpha \in (0, 1)$, find the edge set $A \subseteq E$ with minimum cardinality such that for all traffic matrices T that are routable in E, $\alpha \cdot T$ is routable in A.

Throughout this paper, when mentioning a problem's name or the corresponding abbreviation in sans-serif typeface, e.g. MMCFS, we refer to the optimization question. To indicate that a subgraph is an optimal solution for the problem, we will denote it by the abbreviation in normal typeface, e.g. an MMCFS (V, A) with $A \subseteq E$.

Our contribution. We present the MMCFS problem, which has both a natural formulation and practical applicability. After discussing related problems from literature in Section 2, we give some structural results in Section 3: We show how the MMCFS problem, even though it

is traffic-oblivious in nature, can be reformulated to consider a specific single "hardest" traffic matrix. We also establish how an MCF can be routed in an optimal MMCFS solution, and how the ratio between the values of a feasible MMCFS solution and an optimal one relates to their average edge capacities. In Section 4, we prove that MMCFS is NP-hard already with unit edge capacities. Additionally, we show that it is NP-hard (and a closely related problem cannot be approximated within a sublogarithmic factor) already on directed acyclic graphs (DAGs). Our most important contribution is given in Section 5, where we present a $\max(1/\alpha, 2)$-approximation algorithm for MMCFS: after modelling MMCFS as an ILP, we can deduce a surprisingly simple LP-rounding scheme, whose complexity is solely shifted to the correctness proof. Moreover, we show that our analysis of this algorithm is tight.

2 Related Work

There is a rich body of work on multi-commodity flows – see e.g. [3, Ch. 17] for a primer on this topic and [38] for a recent literature review. The ability to route an MCF in an MMCFS solution not only determines the latter's feasibility, the problem of routing an MCF also has many close ties to several other network design problems. These, however, involve constraints unrelated to MMCFS, are usually not traffic-oblivious, and mostly focus on undirected graphs. Concerning approaches on directed graphs, Foulds [20] minimizes the cost of an MCF in a bidirected network where the use of some unidirectional arcs is prohibited to reduce congestion. Gendron et al. [22, 23] discuss a directed MCF problem that considers costs for both the installation of an edge and the amount of flow routed over it. Further, in *buy-at-bulk network design* [8, 39, 13], capacity on edges must be bought as cheaply as possible such that a given traffic matrix becomes routable – with the caveat that larger amounts of capacity can be bought at a lower price per capacity unit.

In *robust network design* [10, 4, 9, 24], possible traffic matrices are given as an uncertainty set in the form of a polytope, and the objective is usually to minimize the cost of reserving capacity on the edges. While the *dynamic routing* variant considers a different MCF for every traffic matrix in the polytope, *static routing* specifies a fixed unit flow for each commodity that is only scaled with the respective demand value. For directed graphs, Al-Najjar et al. [5] show that an exact algorithm for static routing would yield an $\mathcal{O}(|V|)$-approximation for dynamic routing. Our result can be seen as a better approximation ratio for the special uncertainty set $\{\alpha \cdot T \mid \alpha \in (0, 1),\ T \text{ is routable in } G\}$ under dynamic routing, but w.r.t. minimizing the number of edges in a subgraph rather than the cost of reserved capacity.

There are also several related graph construction and subgraph minimization problems: Khuller et al. [28] give an approximation algorithm for the construction of an undirected tree with constant degree that accommodates given traffic demands between its leaves such that the maximum load on any edge is minimized. Otten et al. [34] evaluate an integer linear program (ILP) and a heuristic for a green traffic engineering problem on digraphs – however, there a specific traffic matrix is also given (and rather than finding an edge subset of minimum cardinality, they minimize the number of "line cards", i.e., sets of 8 incident edges at each vertex). Another well-known topic in the realm of subgraph minimization problems is that of *spanners* [1], i.e., subgraphs that preserve the length of a shortest path within a given ratio (stretch factor) between each pair of vertices. There exists a correspondence between upper bounds on the stretch of shortest paths and the congestion of MCFs, however, this only applies to the existence of probabilistic mappings in undirected graphs [6, 36]. Nonetheless, this correspondence was used to find *flow sparsifiers* in undirected graphs G [18]. While a

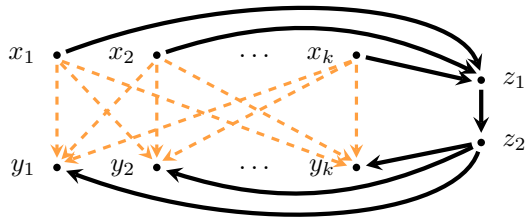


Figure 1 Digraph G constructed in the proof of Observation 1. Edges of the unique optimal MCPS-solution E' of (G, cap, β) are solid (black), and the remaining edges E_{XY} are dashed (orange).

MMCFS solution is similar to a flow sparsifier that preserves the congestion up to a factor $\frac{1}{\alpha}$, they differ in that a flow sparsifier is an entirely new (undirected) graph, not necessarily subgraph, containing a subset of the vertices of G but both old and new edges [32, 30, 7].

Closely related to MMCFS are classical Directed Survivable Network Design (DSND) problems, where, given a (possibly capacitated) input digraph $G = (V, E)$ and a requirement function $r\colon V^2 \to \mathbb{N}$, one aims to find an edge-wise minimum subgraph of G in which one can send a flow of value $r(s,t)$ from s to t for every $(s,t) \in V^2$. On undirected graphs with unit edge capacities, there exists a 2-approximation by Jain [26], which has been adapted to directed instances, but only for a very restricted set of requirement functions [31]. The DSND problem most similar to MMCFS is the Minimum Capacity-Preserving Subgraph (MCPS) problem [15], where the requirement value for a vertex pair (s,t) equals a fraction β of the value of a maximum flow from s to t. However, in all of the aforementioned DSND approaches, each routed commodity is considered in isolation from the others, whereas in the MMCFS setting, all commodities are routed simultaneously. For example, given a digraph $G = (V, E)$ with edge capacities cap and a traffic matrix T routable in E, the scaled-down matrix $\alpha \cdot T$ is not necessarily routable in any optimal MCPS-solution of (G, cap, β) since some edges may be congested by the simultaneous routing of multiple commodities. This is especially easy to see for $\alpha \geq \beta$ but even holds true when $\alpha \ll \beta$:

▶ **Observation 1.** *Given an arbitrarily small $\alpha \in (0,1)$ and an arbitrarily large $\beta \in (0,1)$, there exists a digraph $G = (V, E)$ with edge capacities cap and a traffic matrix T such that T is routable in E, but $\alpha \cdot T$ is not routable in any optimal MCPS-solution E' of (G, cap, β).*

Proof. Let $C := \left\lceil \frac{2\beta}{1-\beta} \right\rceil$ be a (high) edge capacity value and $k > \sqrt{\frac{C}{\alpha}}$ a number of vertices. Construct $G = (V, E)$ (visualized in Figure 1) as follows: Create two vertex sets X and Y with k vertices each, as well as two distinct vertices z_1, z_2, and let $V := X \cup Y \cup \{z_1, z_2\}$. Moreover, let $E_{XY} := X \times Y$, $E' := (X \times \{z_1\}) \cup \{z_1 z_2\} \cup (\{z_2\} \times Y)$, and $E := E_{XY} \cup E'$. Edge capacities and the traffic matrix are chosen as follows:

$$cap(uv) = \begin{cases} 1 & \text{if } uv \in E_{XY}, \\ C & \text{otherwise;} \end{cases} \qquad T(u,v) = \begin{cases} 1 & \text{if } uv \in E_{XY}, \\ 0 & \text{otherwise.} \end{cases}$$

Every non-zero demand $T(u,v)$ can be routed in (G, cap) using the respective edge $uv \in E_{XY}$.

The only optimal MCPS-solution of (G, cap, β) is E': For every vertex pair $(u,v) \in E'$, the only u-v-path in G is the one consisting of the edge uv – so the edges E' must be in the solution. E' also establishes a maximum flow of sufficient value for the remaining vertex pairs. In particular, for every vertex pair $(u,v) \in X \times Y$, there exists a maximum u-v-flow of value C in E', which is at least β times the value $(C+1)$ of a maximum u-v-flow in E:

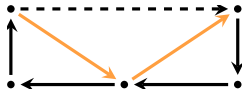


Figure 2 An MMCFS instance with an optimal solution (given by the solid edges) that does not contain the MED. All edge capacities are 1, and $\alpha = \frac{1}{2}$. The MED, which is unique in this example and drawn in black, is not a feasible MMCFS solution: for T with $T(s,t) = cap(st)$ if $e = st \in E$ and 0 otherwise, the dashed edge would have to accommodate a flow of $\frac{3}{2}$ to satisfy all demands, but it only has a capacity of 1.

$$C = \left\lceil \frac{2\beta}{1-\beta} \right\rceil \geq \beta \cdot \frac{2 \cdot (1+\beta-\beta)}{1-\beta} = \beta \left(\frac{2\beta}{1-\beta} + 2 \right) \geq \beta \left(\left\lceil \frac{2\beta}{1-\beta} \right\rceil + 1 \right) = \beta \cdot (C+1)$$

However, the scaled matrix $\alpha \cdot T$ is not routable in E': $|X| \cdot |Y| = k^2 > \frac{C}{\alpha}$ many demands of value α would have to be routed over the single edge $z_1 z_2$, exceeding its capacity C. ◀

Lastly, we want to highlight similarities of MMCFS to the well-established NP-hard Minimum Equivalent Digraph (MED) problem [2, 37, 21, 27], where, given a digraph G, one asks for the edge-wise minimum subgraph of G that preserves the reachability relation of G. In Section 4, we show that MED is a special case of MMCFS. Further, we observe:

▶ **Observation 2.** *In a simple DAG G, the unique minimum equivalent digraph (MED) of G must be contained in every feasible MMCFS solution of G, regardless of edge capacities and α.*

Proof. In any simple DAG G, the MED is unique and consists of exactly those edges st for which there is no s-t-path in $G - st$ [2]. A feasible MMCFS solution must also contain these edges st in order to allow for a non-zero amount of flow from s to t. ◀

However, in general digraphs, a feasible MMCFS solution may not always contain the MED, see Figure 2. Note that MED is not only polynomial-time solvable on DAGs, but there are also several polynomial approximation algorithms for general graphs [29, 45] with the currently best approximation ratio being 1.5 [11, 42].

3 Structural Results

We present some structural insights concerning MMCFS that give a deeper understanding of the problem. Most importantly, we give a reformulation of MMCFS that is used throughout the rest of the paper to obtain structural and algorithmic results: Recall that a feasible solution A for a given MMCFS instance $(G = (V, E), cap, \alpha)$ is defined as an edge set $A \subseteq E$ such that for *all* traffic matrices T that are routable in E, $\alpha \cdot T$ is routable in A. Interestingly, instead of explicitly considering all routable traffic matrices T, it suffices to consider the single specific traffic matrix $\mathbb{T}_E$, which forces each edge to be utilized to its full capacity but has no demands between non-adjacent vertices:

$$\mathbb{T}_E(s,t) := \begin{cases} cap(e) & \text{if } e = st \in E, \\ 0 & \text{otherwise.} \end{cases}$$

We show that an edge set $A \subseteq E$ is a feasible MMCFS solution iff $\alpha \cdot \mathbb{T}_E$ is routable in A:

▶ **Theorem 3.** *Given a digraph $G = (V, E)$ with edge capacities cap, a retention ratio $\alpha \in (0, 1)$, and an edge set $A \subseteq E$, the following statements are equivalent:*

- *For all traffic matrices T that are routable in E, the scaled matrix $\alpha \cdot T$ is routable in A.*
- *The scaled matrix $\alpha \cdot \mathbb{T}_E$ is routable in A.*

Proof. $\mathbb{T}_E$ is routable in E by definition. If every traffic matrix routable in E is also routable in A when scaled down by α, then so is $\alpha \cdot \mathbb{T}_E$. For the other direction, consider any arbitrary traffic matrix T routable in E. Let $\{f^T_{s,t} \mid (s,t) \in V^2\}$ be the MCF that routes T in E with the vector $\mathbf{f}^T \coloneqq \sum_{(s,t)\in V^2} f^T_{s,t}$ specifying the total flow over each edge. Using this MCF, we can construct a new traffic matrix T':

$$T'(s,t) \coloneqq \begin{cases} \mathbf{f}^T(st) & \text{if } e = st \in E, \\ 0 & \text{otherwise.} \end{cases}$$

Observe that $T' \leq \mathbb{T}_E$ when using component-wise comparison. Thus, since $\alpha \cdot \mathbb{T}_E$ is routable in A, so is $\alpha \cdot T'$. But if $\alpha \cdot T'$ is routable in A using the flows $\{f^{\alpha \cdot T'}_{u,v} \mid (u,v) \in V^2\}$, then $\alpha \cdot T$ is also routable in A using the flows $\{f^{\alpha \cdot T}_{s,t} \mid (s,t) \in V^2\}$ constructed as follows: for each commodity $st \in E$ and each edge $uv \in E$, calculate the fraction of flow routed over uv that is used by $f^T_{s,t}$, and route this fraction over the path chosen by $f^{\alpha \cdot T'}_{u,v}$. In short,

$$f^{\alpha \cdot T}_{s,t}(e) \coloneqq \sum_{uv \in E:\ \mathbf{f}^T(uv) > 0} \frac{f^T_{s,t}(uv)}{\mathbf{f}^T(uv)} \cdot f^{\alpha \cdot T'}_{u,v}(e). \qquad \blacktriangleleft$$

We note that a related but slightly different concept to Theorem 3 has been implicitly used previously in [35] (on undirected graphs). From this point onwards, we may refer to edges as commodities since $\mathbb{T}_E$ specifies a non-zero demand precisely for the edges in E. Moreover, given a flow $f_{\tilde{e}}$ for a commodity $\tilde{e} \in E$, we call $f_{\tilde{e}}(\tilde{e})$ the *direct flow* for $\tilde{e}$. We observe that in an optimal MMCFS solution A, for every commodity $\tilde{e} \in A$, the demand $\mathbb{T}_E(\tilde{e})$ can always be fully satisfied by a direct flow $f_{\tilde{e}}(\tilde{e})$:

▶ **Observation 4.** *Let $G = (V, E)$ be a digraph with edge capacities cap and T a traffic matrix routable in an edge set $A \subseteq E$ with $T(e) \leq cap(e)$ for all edges $e \in E$. Then, there exists an MCF $\mathcal{F} = \{f_{s,t} \mid (s,t) \in V^2\}$ in the graph $G' \coloneqq (V, A)$ satisfying the demands T such that $f_{\tilde{e}}(\tilde{e}) = T(\tilde{e})$ for all edges $\tilde{e} \in A$.*

Proof. Among all MCFs that witness the routability of T in A, let $\mathcal{F}' = \{f'_{s,t} \mid (s,t) \in V^2\}$ be one with a maximum sum of direct flow values $\sum_{\tilde{e}\in E} f'_{\tilde{e}}(\tilde{e})$.

We give a proof by contradiction: Assume that there exists an edge $e' = uv \in A$ such that $f'_{e'}(e') < T(e')$ (if no such edge exists, $\mathcal{F} = \mathcal{F}'$ and we are done). There must exist at least one alternative u-v-path P that routes at least some of the remaining demand $T(e') - f'_{e'}(e')$. Further, the edge e' has residual capacity $cap(e') - \sum_{(s,t)\in V^2:} f'_{s,t}(e') = 0$ as otherwise we could increase $f_{e'}(e')$ (and decrease flow along P accordingly), which would contradict the selection of $\mathcal{F}'$. Thus, there exists an edge $e'' \in E$, $e'' \neq e'$, with $f_{e''}(e') > 0$. But then, we can exchange a non-zero amount $\varepsilon > 0$ of flow of commodity e' routed over P with an equally small amount of flow of commodity e'' routed over e'. This increases $f'_{e'}(e')$ without decreasing any other direct flow value – again a contradiction to the selection of $\mathcal{F}'$. $\blacktriangleleft$

Further, for any edge set A, we can compare its total edge capacity $\sum_{e\in A} cap(e)$ to the total flow needed to satisfy the demands $\mathbb{T}_E$. This not only gives us a necessary condition for an edge set A to be a feasible MMCFS solution, but, upon closer analysis, also allows us to relate its quality as a solution to its *mean capacity* $\overline{cap_A} \coloneqq \frac{1}{|A|} \cdot \sum_{e\in A} cap(e)$. The following results apply both in the case of simple and non-simple graphs, but we can give better guarantees in the former case.

▶ **Theorem 5.** *Let O be an optimal solution and $A \neq O$ a feasible solution for an MMCFS instance (G, cap, α). Then, $\frac{|A|}{|O|} \leq \min\left\{\left(1+\frac{1-\alpha}{\theta\alpha}\right)\cdot\frac{\overline{cap_O}}{\overline{cap_A}}, 1+\frac{1-\alpha}{\theta\alpha}\cdot\frac{\overline{cap_O}}{\overline{cap_{A\setminus O}}}\right\}$ with $\theta = 2$ if G has no parallel edges and $\theta = 1$ otherwise.*

Proof. Let $X \coloneqq A \setminus O$, and $Y \coloneqq A \cap O$. The commodities of all $\tilde{e} \in O$ must be routed through O, requiring a total flow of at least $\alpha \sum_{e\in O} cap(e)$ and leaving a total remaining capacity in O of at most $(1-\alpha)\sum_{e\in O} cap(e)$. Every commodity $\tilde{e}' \in X$ has to be routed within this remaining capacity since O is feasible. This requires a total flow of at least $\theta\alpha \sum_{e\in X} cap(e)$; the θ is due to the fact that without parallel edges each such commodity $\tilde{e}'$ must be routed over at least two other edges in O since $\tilde{e}' \notin O$. We thus have

$$\theta\alpha \sum_{e\in X} cap(e) \leq (1-\alpha)\sum_{e\in O} cap(e)\ . \tag{1}$$

By adding $\theta\alpha\sum_{e\in Y} cap(e) \leq \theta\alpha\sum_{e\in O} cap(e)$ to this inequality, we obtain

$$\begin{aligned}
\theta\alpha \sum_{e\in A} cap(e) &\leq (1-\alpha)\sum_{e\in O} cap(e) + \theta\alpha\sum_{e\in O} cap(e)\\
\theta\alpha\cdot|A|\cdot\overline{cap_A} &\leq (1-\alpha+\theta\alpha)\cdot|O|\cdot\overline{cap_O}\\
\frac{|A|}{|O|} &\leq \left(1+\frac{1-\alpha}{\theta\alpha}\right)\cdot\frac{\overline{cap_O}}{\overline{cap_A}}\ .
\end{aligned}$$

Alternatively, we can rewrite inequality (1) as $\theta\alpha\cdot|X|\cdot\overline{cap_X} \leq (1-\alpha)\cdot|O|\cdot\overline{cap_O}$ and obtain

$$\frac{|A|}{|O|} \leq \frac{|O|+|X|}{|O|} = 1+\frac{|X|}{|O|} \leq 1+\frac{1-\alpha}{\theta\alpha}\cdot\frac{\overline{cap_O}}{\overline{cap_X}}\ . \qquad ◀$$

▶ **Corollary 6.** *Any arbitrary feasible solution for MMCFS (including the trivial one, E itself), is a $(1+\frac{1-\alpha}{\theta\alpha}\cdot\frac{\max_{e\in E} cap(e)}{\min_{e\in E} cap(e)})$-approximation.*

This ratio is met, e.g., on a bundle of parallel edges with $\ell \coloneqq (\frac{1}{\alpha}-1)\cdot k$ capacity-1 edges and one capacity-k edge (for any given $\alpha \in \mathbb{Q}_{(0,1)}$ and an arbitrary $k \in \mathbb{Q}$ s.t. $\ell \in \mathbb{N}$).

▶ **Corollary 7.** *For uniform capacities, any arbitrary feasible solution for MMCFS (including the trivial one, E itself) is a $(1+\frac{1-\alpha}{\theta\alpha})$-approximation.*

4 Complexity

Given that even trivial MMCFS solutions satisfy an approximation guarantee according to Corollary 6, one might expect MMCFS to be polynomial-time solvable. However, in this section, we show that MMCFS is NP-hard already on DAGs and give a first inapproximability result. We begin by proving that MMCFS is NP-hard already with unit edge capacities using a reduction from MED that directly follows from Theorem 3:

▶ **Corollary 8.** *MED is the special case of MMCFS with unit edge capacities cap and $\alpha \leq \frac{1}{|E|}$.*

Proof. An optimal solution $A \subseteq E$ for an MMCFS instance (G, cap, α) with unit edge capacities is an edge set of minimum cardinality such that the demands $\alpha\cdot\mathbb{T}_E(s,t) = \alpha\cdot cap(st) \leq \frac{1}{|E|}$ for each edge $st \in E$ are routable in A. This is equivalent to ensuring that there exists an s-t-path in A for every edge $st \in E$ since the (unit) capacity of an edge can never be surpassed by the $|E|$ many flows of size at most $\frac{1}{|E|}$ each. ◀

Moreover, we can show that MMCFS is NP-hard already on DAGs using a reduction from the NP-hard decision variant of Set Cover [27], where one asks: given a universe U, a family of sets $\mathcal{S} = \{S_i \subseteq U\}_{1 \leq i \leq k}$ with $k \in \mathcal{O}(\text{poly}(|U|))$, and a parameter φ, is there a subfamily $\mathcal{C} \subseteq \mathcal{S}$ of cardinality $|\mathcal{C}| \leq \varphi$ such that $\bigcup_{S \in \mathcal{C}} S = U$? The reduction is similar to that given in [15] to prove the NP-hardness of the Minimum Capacity-Preserving Subgraph problem.

▶ **Theorem 9.** *For any fixed $\alpha \in (0,1)$, MMCFS is NP-hard already on DAGs where the longest path has length 3.*

Proof. Given a Set Cover instance $(U, \mathcal{S}, \varphi)$ and a fixed retention ratio $\alpha \in (0,1)$, we construct an instance $I = (G = (V, E), \mathit{cap}, \alpha, \psi)$ for the decision variant of MMCFS: I is a yes-instance if and only if there exists a feasible MMCFS solution $E' \subseteq E$ for $(G, \mathit{cap}, \alpha)$ with cardinality $|E'| \leq \psi$. We construct I as follows (see Figure 3 for a visualization):

$$
\begin{aligned}
V &\coloneqq V_U \cup V_{\mathcal{S}} \cup V_t^{\mathcal{S}} \cup \{t\} \text{ and } E \coloneqq E_U \cup E_{\mathcal{S}} \cup E_1 \\
V_U &\coloneqq \{v_u, v'_u \mid \forall u \in U\} \qquad V_{\mathcal{S}} \coloneqq \{v_S \mid \forall S \in \mathcal{S}\} \qquad V_{\mathcal{S}}^t \coloneqq \{z_S \mid \forall S \in \mathcal{S}\} \\
E_U &\coloneqq V_U \times \{t\} \qquad E_{\mathcal{S}} \coloneqq V_{\mathcal{S}} \times \{t\} \\
E_1 &\coloneqq \{v_u v_S, v'_u v_S \mid \forall S \in \mathcal{S}, u \in S\} \cup \{v_S z_S, z_S t \mid s \in \mathcal{S}\} \\
\mathit{cap}(e) &\coloneqq \begin{cases} 1 \text{ if } e \in E_1; & \frac{1-\alpha}{\alpha} \text{ if } e \in E_{\mathcal{S}}; \qquad \varepsilon \text{ if } e \in E_U, \text{ with } \varepsilon \leq \min\left(\frac{1-\alpha}{\alpha^2 \cdot |U|}, \frac{1-\alpha}{\alpha}\right). \end{cases} \\
\psi &\coloneqq |E_1| + \varphi = 2 \cdot \textstyle\sum_{S \in \mathcal{S}} |S| + 2 \cdot |\mathcal{S}| + \varphi
\end{aligned}
$$

As G is a DAG, its MED is unique [2] and must be part of any feasible MMCFS solution, see Observation 2. This MED is formed by the edges $e \in E_1$; a flow of α must be routed over each of them in order to satisfy the demands $\mathbb{T}_E(e) = \alpha \cdot \mathit{cap}(e)$. The remaining capacity for each edge $e \in E_1$ is $1 - \alpha$.

So consider a single set $S \in \mathcal{S}$ and the corresponding two-path $\{v_S z_S, z_S t\} \in E_1$, whose remaining capacity can thus accommodate either arbitrarily many *item commodities* $\tilde{e}_U \in E_U$ (each one with the sufficiently small demand $\mathbb{T}_E(\tilde{e}_U) = \alpha \cdot \varepsilon$) or a single *set commodity* $\tilde{e}_{\mathcal{S}} \in E_{\mathcal{S}}$ (with the demand $\alpha \cdot \mathbb{T}_E(\tilde{e}_{\mathcal{S}}) = \alpha \cdot \frac{1-\alpha}{\alpha} = 1 - \alpha$). In the former case, we can remove at least two *corresponding item edges* $v_u t$, $v'_u t$ with $u \in S$, which is more than the single *corresponding set edge* $v_S t$ we can remove in the latter case. Thus, for each item commodity $v_u t \in E_U$, an optimal MMCFS solution must contain one of the corresponding set edges $\{v_S t \mid S \ni u\} \subseteq E_{\mathcal{S}}$; the item commodity can then be routed over the path from v_u over v_S to t.

Given a Set Cover solution $\mathcal{C}$ with $|\mathcal{C}| \leq \varphi$, we can construct an MMCFS solution $E' = E_1 \cup \{v_S t \in E_{\mathcal{S}} \mid S \in \mathcal{C}\}$ with cardinality $|E'| = |E_1| + |\mathcal{C}| \leq |E_1| + \varphi = \psi$. Since every item is covered by the sets in $\mathcal{C}$, the constructed MMCFS solution includes at least one corresponding set edge for each item, ensuring its feasibility. Conversely, since a feasible MMCFS solution E' with $|E'| \leq \psi$ has at least one corresponding set edge for each item $u \in U$, the Set Cover solution $\mathcal{C} = \{S \mid v_S t \in E_{\mathcal{S}} \cap E'\}$ also contains at least one covering set for each item. Moreover, $|\mathcal{C}| = |E'| - |E_1| \leq \psi - |E_1| = \varphi$. ◀

Considering the optimization variants of Set Cover and MMCFS (and thus ignoring the additional input values φ and ψ), the reduction above also implies the inapproximability of the number of edges in an optimal MMCFS solution beyond the edges required for an MED: Consider an instance $I = (G = (V, E), \mathit{cap}, \alpha)$ for the optimization variant of MMCFS that is produced by the reduction above, and an arbitrary feasible solution $A \subseteq E$ for this instance. Let $\mu(I, A) \coloneqq |A| - \mathit{med}(G)$ where $\mathit{med}(G)$ denotes the number of edges in an MED of G. Then, A can be transformed into a feasible solution for the original Set Cover instance with objective value $\mu(I, A)$ in linear time. Further, let $\mu(I)$ be the minimum $\mu(I, A')$ over all

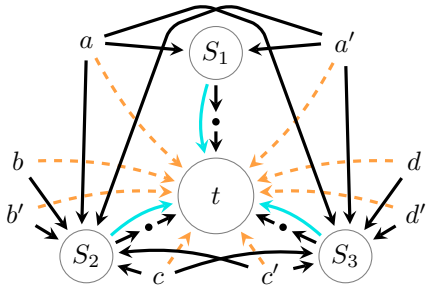


Figure 3 MMCFS instance constructed from a Set Cover instance with universe $U = \{a, b, c, d\}$ and family of subsets $\mathcal{S} = \{\{a\},\ \{a, b, c\},\ \{a, c, d\}\}$. An optimal solution contains the (solid black) MED as well as one (blue) corresponding set edge for each $u \in U$. Item edges are orange and dashed.

feasible solutions A' for I, and recall that the size $|E|$ of the MMCFS instance I is linear in the size $N \in \mathcal{O}(\text{poly}(U))$ of the Set Cover instance: if it was possible to approximate $\mu(I)$ within a factor in $o(\log |E|) = o(\log |U|)$, one could also approximate Set Cover within $o(\log |U|)$, which is NP-hard [17, 33]. This implies that any approximation algorithm for MMCFS (such as the one we present in Section 5) must leverage the existence of a comparatively high number of MED-edges in order to achieve its approximation ratio.

▶ **Observation 10.** *Given an MMCFS instance $I = (G = (V, E), cap, \alpha)$ and an MED of G, approximating $\mu(I)$ with a ratio in $o(\log |E|)$ is NP-hard. This already holds on DAGs where the longest path has length 3.*

5 LP-based Approximation

We present an extremely simple $\max(1/\alpha, 2)$-approximation for MMCFS based on LP rounding. This is a clear improvement over the default approximation guarantee of Corollary 6, which depends the instance's edge capacities and is thus not even polynomially bounded in α^{-1} or the instance size. Consider the following ILP formulation for MMCFS:

$$\min \sum_{e \in E} x_e \tag{1.a}$$

$$\sum_{u:\ uv \in E} f_{\tilde{e}}(uv) - \sum_{u:\ vu \in E} f_{\tilde{e}}(vu) = \begin{cases} -\alpha \mathbb{T}_E(\tilde{e}) & \text{if } v = s \\ \alpha \mathbb{T}_E(\tilde{e}) & \text{if } v = t \\ 0 & \text{else} \end{cases} \qquad \forall v \in V, \tilde{e} = st \in E \tag{1.b}$$

$$\sum_{\tilde{e} \in E} f_{\tilde{e}}(e) \leq x_e \cdot cap(e) \qquad \forall e \in E \tag{1.c}$$

$$f_{\tilde{e}}(e) \geq 0 \qquad \forall e \in E, \tilde{e} \in E \tag{1.d}$$

$$x_e \in \{0, 1\} \qquad \forall e \in E \tag{1.e}$$

A binary indicator variable x_e determines whether edge $e \in E$ is part of the solution subgraph or not. We minimize the sum of these variables in the objective function (1.a) to obtain a subgraph of minimum size. The non-negative variables $f_{\tilde{e}}(e)$ determine the amount of flow routed over edge e for commodity $\tilde{e} \in E$. Recall that $\mathbb{T}_E$ specifies a demand of $cap(\tilde{e})$ precisely for each edge $\tilde{e}$: the flow preservation constraints (1.b) guarantee that the f-variables represent proper flows that satisfy these demands. Lastly, the capacity constraints (1.c) ensure that the total sum of flow over any edge $e \in E$ does not surpass the capacity of e. This flow sum $\mathbf{f}(e) \coloneqq \sum_{\tilde{e} \in E} f_{\tilde{e}}(e)$ must be zero if e is not part of the solution.

We obtain the relaxation of ILP (1) by replacing the integrality constraints (1.e) on x_e by the inequalities $0 \leq x_e \leq 1$ for all $e \in E$. The only other type of constraint that bounds the x_e-variables is (1.c). Since the x_e-variables can assume fractional values in the LP relaxation, and the sum over all x_e is minimized, this lower bound of $\frac{\mathbf{f}(e)}{cap(e)}$ on x_e for all $e \in E$ will always be met with equality. Hence, the LP relaxation is equivalent to a standard MCF-LP that minimizes the sum of edge utilizations in the objective function $\sum_{e \in E} \frac{\mathbf{f}(e)}{cap(e)}$. Accordingly, we will refer to $cost(e) \coloneqq \frac{1}{cap(e)}$ as the *cost* that routing a single unit of flow over an edge e will add to this objective.

▶ Remark 11. The solution $x_e = \alpha$ for all $e \in E$ with objective value $\alpha|E|$ is always feasible for the relaxation of ILP (1). When the input graph is simple and all edge capacities are uniform, it is in fact optimal: the flow $f_{\tilde{e}}$ for commodity $\tilde{e}$ will always be routed completely over $\tilde{e}$ since all edges have the same cost and routing $f_{\tilde{e}}$ over an alternative path would cost more than routing it over a single edge. This implies that any arbitrary feasible solution for MMCFS on simple graphs with uniform edge capacities is an $(1/\alpha)$-approximation (which we already proved using a separate argument via Corollary 7). However, below we also use the LP relaxation to approximate MMCFS on non-simple graphs with non-uniform edge capacities.

For our approximation algorithm, we make use of the well-known fact that all standard LP solving algorithms will always return a *basic* optimal solution (if any solution exists) [43, p. 279]. A basic optimal solution – or equivalently, extreme point solution – is a vertex of the polyhedron defined as the convex hull of the set of feasible solutions. It does not lie on a higher-dimensional face of the polyhedron and thus cannot be expressed as a convex combination of two or more other feasible solutions [41, p. 100]. We prove that a basic optimal solution for the relaxation of ILP (1) will only have comparatively few variables with a positive value below α, allowing us to round up the solution to obtain an approximation.

▶ **Lemma 12.** *Let x^* be a basic optimal solution for the relaxation of ILP (1), and h the number of edges e such that $x^*_e = 1$. There exist at most h many edges e' with $x^*_{e'} \in (0, \alpha)$.*

Proof. Let $H \coloneqq \{e \in E \mid x^*_e = 1\}$ be the h many edges that are saturated by the fractional multi-commodity flow, and $L \coloneqq \{e \in E \mid x^*_e \in (0, \alpha)\}$ with $\ell \coloneqq |L|$ the edges that are only used to a fraction less than α; in short, edges with *high* and *low* corresponding x^*-values. We show that an optimal fractional solution x^* with $\ell > h$ would allow us to construct a vector $p \in \mathbb{Q}^{|E|}$ such that both $(x^* + p)$ and $(x^* - p)$ are still feasible solutions for the LP relaxation – thus, x^* would not be basic.

For every edge $e = st \in L$, let P_e denote an arbitrary *alternative s-t-path* not using the edge e and $f_e(e') > 0$ for all edges $e' \in P_e$. Such a path must exist since at most $x^*_e \cdot cap(e) < \alpha \cdot cap(e)$ flow is routed over e, so an alternative s-t-path is necessary to satisfy the demand $\alpha \cdot \mathbb{T}_E(s,t) = \alpha \cdot cap(e)$. Further, the total cost of routing a unit of flow over P_e must be lower than or equal to the cost of routing it over e itself, i.e., $\sum_{e' \in P_e} \frac{1}{cap(e')} \leq \frac{1}{cap(e)}$, by the optimality of x^*. Based on the alternative paths, we construct a matrix $M \in \{0,1\}^{\ell \times h}$ indexed by pairs $(e, e'') \in L \times H$, with $M(e, e'') \coloneqq 1$ if $e'' \in P_e$, and 0 otherwise. Since $\ell > h$, the ℓ rows of M must be linearly dependent, i.e., there exists a vector of coefficients $q \in \mathbb{Q}^\ell$ with $q \neq \mathbf{0}$, such that $q^\top \cdot M = \mathbf{0}$ (and consequently, $-q^\top \cdot M = \mathbf{0}$).

We can obtain two new feasible solutions by modifying the MCF corresponding to the optimal basic solution x^* as follows: for each edge $e \in L$, and using a small positive value $\varepsilon \in \mathbb{Q}$, we send $\varepsilon \cdot q(e)$ less flow over e itself and $\varepsilon \cdot q(e)$ more flow over the alternative path P_e – or vice versa. Put formally, we argue that for a small enough $\varepsilon > 0$, the following vector $p \in \mathbb{Q}^{|E|}$ yields two feasible solutions $(x^* + p)$ and $(x^* - p)$ for the LP relaxation:

$$p(e') = \begin{cases} \varepsilon \cdot q(e') - \varepsilon \cdot \sum_{e \in L\colon e' \in P_e} q(e) & \text{if } e' \in L, \\ -\varepsilon \cdot \sum_{e \in L\colon e' \in P_e} q(e) & \text{otherwise.} \end{cases}$$

Clearly, the two solutions satisfy all demands and flow preservation constraints (1.b). Consider the capacity constraints (1.c): By construction of q, the flow difference on saturated edges is 0. For all non-saturated edges e' it holds: if we modified the flow routed over e', this flow was already non-zero before our modification, and ε can be chosen sufficiently small such that the flow will neither turn negative nor surpass $cap(e')$.

It remains to show that $p \neq \mathbf{0}$ given that $q \neq \mathbf{0}$. So, among the edges $e \in L$ with $q(e) \neq 0$, choose one with maximum cost $\frac{1}{cap(e)}$ and denote it by $e_{\max}$. We show that $p(e_{\max}) \neq 0$.

If $e_{\max}$ were contained in any of the alternative paths P_e with $e \in L$ and $q(e) \neq 0$, we would arrive at a contradiction even without using p: Recall that $\sum_{e' \in P_e} \frac{1}{cap(e')} \leq \frac{1}{cap(e)}$. So we must have $|P_e| = 1$, i.e., e and $e_{\max}$ are parallel edges with $cap(e) = cap(e_{\max})$. Since $e, e_{\max} \in L$ and thus $x^*_e, x^*_{e_{\max}} \in (0, \alpha)$, we could simply obtain two new solutions by shifting some small $\varepsilon > 0$ of flow from one to another, or vice versa, contradicting that x^* is basic.

Hence, $e_{\max}$ is only contained in alternative paths P_e with $e \in L$ s.t. $q(e) = 0$, and

$$p(e_{\max}) = \varepsilon \cdot q(e_{\max}) - \varepsilon \cdot \sum_{e \in L\colon e_{\max} \in P_e} q(e) = \varepsilon \cdot q(e_{\max}) - 0 \neq 0. \qquad \blacktriangleleft$$

The proof implicitly gives an intuitive explanation for the distribution of LP values:

▶ **Corollary 13.** *Let x^* be a basic optimal solution for the relaxation of ILP (1). For each edge $e = st$ with $x^*_e \in (0, \alpha)$ it holds: every* alternative *s-t-*path *P_e (not containing e) with $f_e(e') > 0$ for all of its edges $e' \in P_e$ must contain an edge e'' with $x^*_{e''} = 1$.*

Proof. Assume that, for some edge $e = st$ with $x^*_e \in (0, \alpha)$, there is a P_e that does not contain an edge e'' with $x^*_{e''} = 1$. Then, we can follow the proof of Lemma 12, choosing P_e as the alternative s-t-path for e. As a result, the matrix M constructed in the proof contains a row of zeroes, allowing us to route ε more flow over e and ε less flow over P_e, or vice versa. Thus, x^* cannot be basic, a contradiction. ◀

We now analyze the following LP-rounding algorithm: compute a basic optimal solution x^* for the relaxation of ILP (1) in polynomial time, and return the edge set $\{e \in E \mid x^*_e > 0\}$. Based on Lemma 12, one could use naïve techniques to prove approximation guarantees of $(2 + \frac{1}{\alpha})$ or $\frac{2}{\alpha}$ for this algorithm. However, we provide the stronger bound of $\max(1/\alpha, 2)$ based on the following intuition: the algorithm either "misses" the optimal solution mainly due to edges with $x^*_e \in (0, \alpha)$ and we can obtain a 2-approximation via Lemma 12, or due to edges with $x^*_e \in [\alpha, 1)$ and we can round up the variables for a $\frac{1}{\alpha}$-approximation.

▶ **Theorem 14.** *Let x^* be a basic optimal solution for the relaxation of ILP (1). The edge set $A := \{e \in E \mid x^*_e > 0\}$ is a $\max(1/\alpha, 2)$-approximation for MMCFS. That is, rounding up x^* is a 2-approximation when $\alpha \geq \frac{1}{2}$, and an $1/\alpha$-approximation when $\alpha < \frac{1}{2}$.*

Proof. The solution A is clearly feasible. Let $z := |A|$ be the number of edges e such that $x^*_e > 0$. Furthermore, let $\Delta = |\{e \in E \mid x^*_e \in [\alpha, 1)\}|$ be the number of these z edges whose x^*-variable is set to a value in the interval $[\alpha, 1)$. We know that the remaining $(z - \Delta)$ edges have a x^*-variable either set to 1 or a value lower than α. In particular, by Lemma 12, there exist at least as many edges e' with $x^*_{e'} = 1$ as there are edges e'' with $x^*_{e''} \in (0, \alpha)$. Thus, $|\{e \in E \mid x^*_e = 1\}| \geq \frac{z - \Delta}{2}$. Hence, the optimal fractional solution x^* has objective value

$$\begin{aligned} z^* &\geq |\{e \in E \mid x^*_e = 1\}| + \alpha \cdot |\{e \in E \mid x^*_e \in [\alpha, 1)\}| \\ &\geq \frac{z - \Delta}{2} + \alpha \cdot \Delta = \left(\frac{1}{2} + (\alpha - \frac{1}{2}) \cdot \frac{\Delta}{z}\right) \cdot z. \end{aligned}$$

Using the minimum fractional objective value z^* as a lower bound for the minimum integral objective value z^*_{int}, we can then give an upper bound for the approximation ratio:

$$r \coloneqq \frac{z}{z^*_{int}} \leq \frac{z}{z^*} \leq \frac{1}{\frac{1}{2} + (\alpha - \frac{1}{2}) \cdot \frac{\Delta}{z}}$$

To obtain an upper bound on this ratio, we examine when its denominator is at its minimum. For $\alpha \geq \frac{1}{2}$, the term $(\alpha - \frac{1}{2}) \cdot \frac{\Delta}{z}$ is always non-negative and reaches its lowest value 0 when $\Delta = 0$, giving us the upper bound $2 \geq r$ in this case. In contrast, for $\alpha < \frac{1}{2}$, the term $(\alpha - \frac{1}{2}) \cdot \frac{\Delta}{z}$ is negative, and the minimum of the denominator is reached when $\Delta = z$, leading to the upper bound $\frac{1}{\alpha} \geq r$ in this second case. ◀

It is surprisingly hard to find instances (in particular, ones without parallel edges) where the worst-case approximation ratio given by Theorem 14 is actually met. However, we show that such instances exist for the most relevant $\alpha \in (0, 1)$, and hence our analysis is *tight*:

▶ **Lemma 15.** *The ratio $1/\alpha$ for the algorithm of Theorem 14 is tight for all $\alpha = \frac{1}{q}$, $q \in \mathbb{N}_{>1}$.*

Proof. Consider a complete bidirected graph $G = (V, E)$ on $q + 1$ vertices, and let $H \subset G$ denote an arbitrarily chosen directed Hamiltonian cycle in G (an example for $q = 4$ is visualized in Figure 4). For all edges $st \in E$, let $\eta(st)$ denote the number of edges of the unique s-t-path in H and set the capacity $cap(st) \coloneqq \frac{1}{\eta(st)}$. $E(H)$ is a feasible integral solution for the MMCFS instance (G, cap, α) since the demands $\alpha \cdot \mathbb{T}_E$ are routable in it: A single edge $e \in H$ is able to satisfy its own demand $\alpha \cdot cap(e) = \frac{1}{q} \cdot 1$. Adding to this, for every $i \in \{2, \dots, q\}$, there are i edges $uv \in E \setminus E(H)$ with $\eta(uv) = i$ whose unique u-v-path in H contains e. The edge e can accommodate all of these commodities uv because each of them has a demand of $\alpha \cdot cap(uv) = \frac{1}{q} \cdot \frac{1}{i}$, summing up to a total flow of $\sum_{i=1}^{q} i \cdot \frac{1}{i} \cdot \frac{1}{q} = 1 \leq \frac{1}{cap(e)}$. Lastly, $E(H)$ is not only feasible but optimal since we need at least $|E(H)|$ many edges to preserve the reachability relation of G in H.

The algorithm from Theorem 14 would potentially choose all edges of G, i.e., $|E| = (|V| - 1) \cdot |V| = q \cdot |E(H)| = \frac{1}{\alpha} \cdot |E(H)|$ many: As the cost $\frac{1}{cap(uv)} = \eta(uv)$ of an edge $uv \in E \setminus E(H)$ is equal to the total cost of its unique u-v-path in H, all u-v-paths are equal in cost. Thus, one optimal solution for the relaxation of ILP (1) simply routes all commodities $\tilde{e} \in E$ completely over their own edges $\tilde{e}$ and chooses $x^*_{\tilde{e}} = \alpha$. This solution is also basic: Let f denote the flow variables of this solution. If it were not basic, there would exist two other optimal solutions with flow variables f', f'' respectively, such that for some $\tilde{e}, e \in E$, $f'_{\tilde{e}}(e) < f_{\tilde{e}}(e) < f''_{\tilde{e}}(e)$. However, if $e = \tilde{e}$, f already routes the full demand of commodity $\tilde{e}$ over e and we cannot increase $f_{\tilde{e}}(e)$ without introducing a flow cycle, which would raise the objective value and thus be suboptimal. In contrast, if $e \neq \tilde{e}$, $f_{\tilde{e}}(e)$ is already 0 and cannot be decreased. ◀

▶ **Lemma 16.** *The ratio 2 for the algorithm of Theorem 14 is tight for all $\alpha > \frac{1}{2}$.*

Proof. We construct a family of MMCFS instances (visualized in Figure 5), each one consisting of a simple graph $G = (V, E)$ with edge capacities cap and the given retention ratio α, that meets the approximation ratio of 2 asymptotically with increasing size. Intuitively, the edge set of each instance contains two subsets that have a size quadratic in the number of vertices and two subsets of linear size; our algorithm chooses both quadratic-size subsets even though one could be replaced by a linear-size subset.

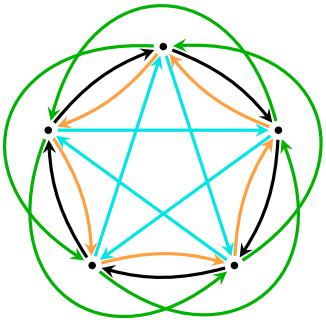


Figure 4 MMCFS instance for $\alpha = \frac{1}{4}$, constructed as a member of the family of instances in the proof of Lemma 15. The Hamiltonian cycle H is drawn as black. The value $\eta(st)$, denoting the distance from s to t in H for an edge st, is 2, 3, or 4 for blue, green, and orange edges, respectively.

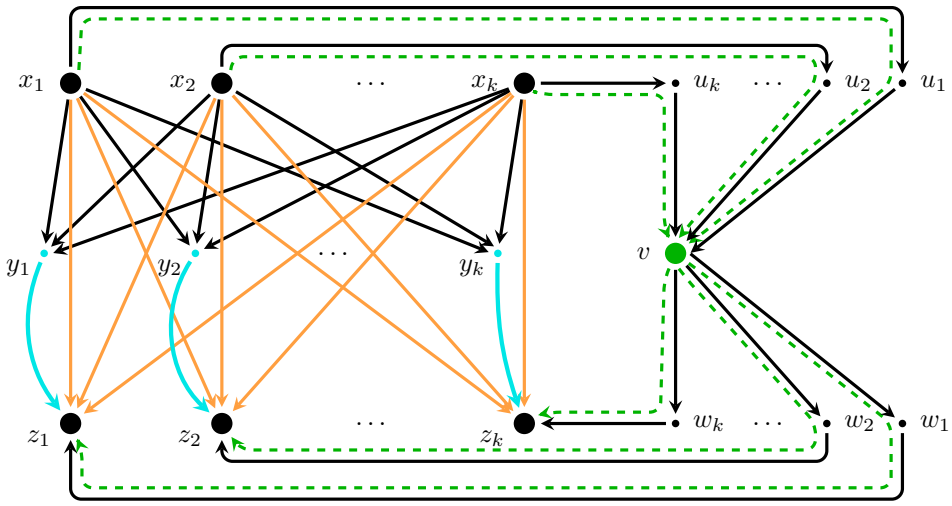


Figure 5 Family of MMCFS instances constructed in the proof of Lemma 16. Edges in E_v are drawn in green (and dashed), edges in E_{XZ} are orange, edges in E_1 are black, and edges in E_B are blue. Recall that edges E_1 and E_B (black and blue) form the unique MED, which is contained in every feasible solution. There exists a feasible MMCFS solution $E_1 \cup E_B \cup E_v$ without any (orange) edges from E_{XZ}, but the algorithm from Theorem 14 will include all edges of E_{XZ} in its solution.

Let V be comprised of five disjoint vertex sets X, Y, Z, U, W with $k \in \mathbb{N}$ vertices each (respectively named with the corresponding lowercase letter and indexed by $i \in \{1, \dots, k\}$) and a distinct vertex v. Further, let $\varepsilon \leq \frac{1-\alpha}{k \cdot \alpha}$ and $B \geq \frac{k \cdot \varepsilon}{1-\alpha}$ be sufficiently low and high positive values, respectively. Then, $E \coloneqq E_v \cup E_{XZ} \cup E_1 \cup E_B$ with

$$E_v \coloneqq (X \times \{v\}) \cup (\{v\} \times Z) \qquad E_{XZ} \coloneqq X \times Z$$
$$E_1 \coloneqq X \times Y \cup \{x_i u_i, u_i v, v w_i, w_i z_i \mid i \in \{1, \dots, k\}\} \qquad E_B \coloneqq \{y_i z_i \mid i \in \{1, \dots, k\}\}$$
$$cap(e) \coloneqq \begin{cases} \frac{1-\alpha}{\alpha} \text{ if } e \in E_v, & \frac{1-\alpha+\varepsilon}{\alpha} \text{ if } e \in E_{XZ}, & 1 \text{ if } e \in E_1, & B \text{ if } e \in E_B. \end{cases}$$

G is a DAG, thus its MED is unique [2] and must be part of any feasible solution for the MMCFS instance (G, cap, α), see Observation 2. The MED of G consists of the edges $E_1 \cup E_B$ and in particular can satisfy all demands $\alpha \cdot \mathbb{T}_E(e) = \alpha \cdot cap(e)$ with $e \in E_1 \cup E_B$. The "remaining" capacity of $1 - \alpha$ for each of the edges in E_1 (and $(1 - \alpha) \cdot B$ for edges in E_B) is also sufficient to satisfy the demands $\alpha \cdot \mathbb{T}_E(e_v)$ of the commodities $e_v \in E_v$, and to *almost* satisfy the demands $\alpha \cdot \mathbb{T}_E(e_{XZ}) = 1 - \alpha + \varepsilon$ for the commodities $e_{XZ} \in E_{XZ}$: it only leaves a demand of ε for every such e_{XZ}. Thus, there exists a feasible solution $E_1 \cup E_B \cup E_v$ for (G, cap, α) that contains all of the $k^2 + 5k$ edges from $E_1 \cup E_B$ and additionally, to satisfy the remaining demands (of size ε each), the $2k$ edges from E_v. This feasible solution gives an upper bound on the objective value z^*_{int} of the optimal integral solution: $z^*_{int} \leq k^2 + 7k$.

In contrast, the algorithm from Theorem 14 also includes all $k^2 + 5k$ edges of the MED $E_1 \cup E_B$ in its solution, but must additionally choose (at least) the k^2 edges from E_{XZ}, resulting in an objective value $z \geq 2k^2 + 5k$. This is because every optimal solution for the relaxation of ILP (1) routes the commodities $st = e_{XZ} \in E_{XZ}$ over the edges e_{XZ} themselves: they have a lower cost than the alternative s-t-path of length 2 over the edges in E_v.

For increasing k, the ratio of the algorithmic solution's objective value to the optimum is

$$\lim_{k \to \infty} \frac{z}{z^*_{int}} \geq \lim_{k \to \infty} \frac{2k^2 + 5k}{k^2 + 7k} = 2. \qquad \blacktriangleleft$$

Combining Theorem 14 with Lemmas 15 and 16, we obtain:

▶ **Corollary 17.** *The approximation ratio* $\max(1/\alpha, 2)$ *for the algorithm given by Theorem 14 is tight for all* $\alpha > \frac{1}{2}$ *and all* $\alpha = \frac{1}{q}$ *with* $q \in \mathbb{N}_{>1}$.

▶ Remark 18. There are instances where the integrality gap of ILP (1) is $\frac{1}{\alpha}$ – e.g. trees, where the unique optimal integral solution contains every edge while the unique optimal fractional solution chooses $x^*_e = \alpha$ for all edges e of the input graph. Interestingly, the approximation ratio of our algorithm (for $\alpha \leq \frac{1}{2}$) equals the integrality gap exactly, but on completely different instances and not on trees (where the algorithm always finds the optimal solution).

6 Conclusion and Open Questions

We introduced the practically motivated Minimum Multi-Commodity Flow Subgraph (MMCFS) problem and paved the way for further research by giving several structural results, most importantly a reformulation of this traffic-oblivious problem that only needs to consider a single specific traffic matrix. Further, we showed that MMCFS is NP-hard (and a closely related problem cannot be approximated within a sublogarithmic factor) already on DAGs. Lastly, we gave an extremely simple LP-rounding scheme for MMCFS with a tight approximation guarantee of $\max(1/\alpha, 2)$.

Considering seemingly related problems (see Section 2), one observes that an approximation ratio of 2 (which we attain for the practically most relevant cases of $\alpha \geq \frac{1}{2}$ [34]) often arises as a seemingly "natural" limit for such ratios. Yet, it remains an open question whether there exists an approximation algorithm for MMCFS with a better quality guarantee, and whether there is a non-trivial lower bound on the approximation guarantee for any such algorithm (assuming P $\neq$ NP). Further, it might be of interest to explore several generalizations of MMCFS: this includes the non-traffic-oblivious variant where a specific traffic matrix is part of the input (which is also NP-hard via Theorem 3), and the variant where, given an additional cost function on the edges, one asks for a subgraph of minimum cost.

References

1 Abu Reyan Ahmed, Greg Bodwin, Faryad Darabi Sahneh, Keaton Hamm, Mohammad Javad Latifi Jebelli, Stephen G. Kobourov, and Richard Spence. Graph spanners: A tutorial review. *Comput. Sci. Rev.*, 37:100253, 2020. doi:10.1016/J.COSREV.2020.100253.

2 Alfred V. Aho, Michael R. Garey, and Jeffrey D. Ullman. The transitive reduction of a directed graph. *SIAM J. Comput.*, 1(2):131–137, 1972. doi:10.1137/0201008.

3 Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows – Theory, algorithms and applications*. Prentice Hall, 1993.

4 Yacine Al-Najjar, Walid Ben-Ameur, and Jérémie Leguay. On the approximability of robust network design. *Theor. Comput. Sci.*, 860:41–50, 2021. doi:10.1016/J.TCS.2021.01.026.

5 Yacine Al-Najjar, Walid Ben-Ameur, and Jérémie Leguay. Approximability of robust network design: The directed case. In *Proc. STACS 2022*, volume 219 of *LIPIcs*, pages 6:1–6:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.STACS.2022.6.

6 Reid Andersen and Uriel Feige. Interchanging distance and capacity in probabilistic mappings. *CoRR*, abs/0907.3631, 2009. arXiv:0907.3631.

7 Alexandr Andoni, Anupam Gupta, and Robert Krauthgamer. Towards $(1 + \epsilon)$-approximate flow sparsifiers. In *Proc. SODA 2014*, pages 279–293. SIAM, 2014. doi:10.1137/1.9781611973402.20.

8 Spyridon Antonakopoulos. Approximating directed buy-at-bulk network design. In *Proc. WAOA 2010*, volume 6534 of *LNCS*, pages 13–24. Springer, 2010. doi:10.1007/978-3-642-18318-8_2.

9 Yossi Azar, Edith Cohen, Amos Fiat, Haim Kaplan, and Harald Räcke. Optimal oblivious routing in polynomial time. *J. Comput. Syst. Sci.*, 69(3):383–394, 2004. doi:10.1016/J.JCSS.2004.04.010.

10 Walid Ben-Ameur and Hervé Kerivin. Routing of uncertain traffic demands. *Optimization and Engineering*, 6:283–313, 2005. doi:10.1145/777313.777314.

11 Piotr Berman, Bhaskar DasGupta, and Marek Karpinski. Approximating transitive reductions for directed networks. In *Proc. WADS 2009*, volume 5664 of *LNCS*, pages 74–85. Springer, 2009. doi:10.1007/978-3-642-03367-4_7.

12 Randeep Bhatia, Fang Hao, Murali S. Kodialam, and T. V. Lakshman. Optimized network traffic engineering using segment routing. In *Proc. INFOCOM 2015*, pages 657–665. IEEE, 2015. doi:10.1109/INFOCOM.2015.7218434.

13 Chandra Chekuri, Mohammad Taghi Hajiaghayi, Guy Kortsarz, and Mohammad R. Salavatipour. Approximation algorithms for nonuniform buy-at-bulk network design. *SIAM J. Comput.*, 39(5):1772–1798, 2010. doi:10.1137/090750317.

14 Luca Chiaraviglio, Marco Mellia, and Fabio Neri. Reducing power consumption in backbone networks. In *Proc. ICC 2009*, pages 1–6. IEEE, 2009. doi:10.1109/ICC.2009.5199404.

15 Markus Chimani and Max Ilsen. Capacity-preserving subgraphs of directed flow networks. In *Proc. IWOCA 2023*, volume 13889 of *LNCS*, pages 160–172. Springer, 2023. doi:10.1007/978-3-031-34347-6_14.

16 Markus Chimani and Max Ilsen. Traffic-oblivious multi-commodity flow network design. *CoRR*, abs/2504.16744, 2025. doi:10.48550/arXiv.2504.16744.

17 Irit Dinur and David Steurer. Analytical approach to parallel repetition. In *Proc. STOC 2014*, pages 624–633. ACM, 2014. doi:10.1145/2591796.2591884.

18 Matthias Englert, Anupam Gupta, Robert Krauthgamer, Harald Räcke, Inbal Talgam-Cohen, and Kunal Talwar. Vertex sparsifiers: New results from old techniques. *SIAM J. Comput.*, 43(4):1239–1262, 2014. doi:10.1137/130908440.

19 Clarence Filsfils, Stefano Previdi, Les Ginsberg, Bruno Decraene, Stephane Litkowski, and Rob Shakir. Segment routing architecture. *RFC*, 8402:1–32, 2018. doi:10.17487/RFC8402.

20 Les R. Foulds. A multi-commodity flow network design problem. *Transportation Research Part B: Methodological*, 15(4):273–283, 1981. doi:10.1016/0191-2615(81)90013-8.

21 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

22 Bernard Gendron, Teodor Gabriel Crainic, and Antonio Frangioni. *Multicommodity Capacitated Network Design*, pages 1–19. Springer US, Boston, MA, 1999. doi:10.1007/978-1-4615-5087-7_1.

23 Bernard Gendron and Mathieu Larose. Branch-and-price-and-cut for large-scale multicommodity capacitated fixed-charge network design. *EURO J. Comput. Optim.*, 2(1-2):55–75, 2014. doi:10.1007/S13675-014-0020-9.

24 Mohammad Taghi Hajiaghayi, Robert D. Kleinberg, Harald Räcke, and Tom Leighton. Oblivious routing on node-capacitated and directed graphs. *ACM Trans. Algorithms*, 3(4):51, 2007. doi:10.1145/1290672.1290688.

25 Renaud Hartert, Pierre Schaus, Stefano Vissicchio, and Olivier Bonaventure. Solving segment routing problems with hybrid constraint programming techniques. In *Proc. CP 2015*, volume 9255 of *LNCS*, pages 592–608. Springer, 2015. doi:10.1007/978-3-319-23219-5_41.

26 Kamal Jain. A factor 2 approximation algorithm for the generalized steiner network problem. *Comb.*, 21(1):39–60, 2001. doi:10.1007/s004930170004.

27 Richard M. Karp. Reducibility among combinatorial problems. In *Proc. COCO 1972*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_9.

28 Samir Khuller, Balaji Raghavachari, and Neal E. Young. Designing multi-commodity flow trees. *Inf. Process. Lett.*, 50(1):49–55, 1994. doi:10.1016/0020-0190(94)90044-2.

29 Samir Khuller, Balaji Raghavachari, and Neal E. Young. Approximating the minimum equivalent digraph. *SIAM J. Comput.*, 24(4):859–872, 1995. doi:10.1137/S0097539793256685.

30 Frank Thomson Leighton and Ankur Moitra. Extensions and limits to vertex sparsification. In *Proc. STOC 2010*, pages 47–56. ACM, 2010. doi:10.1145/1806689.1806698.

31 Vardges Melkonian and Éva Tardos. Algorithms for a network design problem with crossing supermodular demands. *Networks*, 43(4):256–265, 2004. doi:10.1002/NET.20005.

32 Ankur Moitra. Approximation algorithms for multicommodity-type problems with guarantees independent of the graph size. In *Proc. FOCS 2009*, pages 3–12. IEEE Computer Society, 2009. doi:10.1109/FOCS.2009.28.

33 Dana Moshkovitz. The projection games conjecture and the np-hardness of ln n-approximating set-cover. *Theory Comput.*, 11:221–235, 2015. doi:10.4086/toc.2015.v011a007.

34 Daniel Otten, Max Ilsen, Markus Chimani, and Nils Aschenbruck. Green traffic engineering by line card minimization. In *Proc. LCN 2023*, pages 1–8. IEEE, 2023. doi:10.1109/LCN58197.2023.10223344.

35 Harald Räcke. Minimizing congestion in general networks. In *Proc. FOCS 2002*, pages 43–52. IEEE Computer Society, 2002. doi:10.1109/SFCS.2002.1181881.

36 Harald Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *Proc. STOC 2008*, pages 255–264. ACM, 2008. doi:10.1145/1374376.1374415.

37 Sartaj Sahni. Computationally related problems. *SIAM J. Comput.*, 3(4):262–279, 1974. doi:10.1137/0203021.

38 Khodakaram Salimifard and Sara Bigharaz. The multicommodity network flow problem: state of the art classification, applications, and solution methods. *Oper. Res.*, 22(1):1–47, 2022. doi:10.1007/S12351-020-00564-8.

39 F. Sibel Salman, Joseph Cheriyan, R. Ravi, and S. Subramanian. Buy-at-bulk network design: Approximating the single-sink edge installation problem. In *Proc. SODA 1997*, pages 619–628. ACM/SIAM, 1997. URL: http://dl.acm.org/citation.cfm?id=314161.314397.

40 Timmy Schüller, Nils Aschenbruck, Markus Chimani, Martin Horneffer, and Stefan Schnitter. Traffic engineering using segment routing and considering requirements of a carrier IP network. *IEEE/ACM Trans. Netw.*, 26(4):1851–1864, 2018. doi:10.1109/TNET.2018.2854610.

41 Vijay V. Vazirani. *Approximation algorithms*. Springer, 2001. URL: http://www.springer.com/computer/theoretical+computer+science/book/978-3-540-65367-7.

42 Adrian Vetta. Approximating the minimum strongly connected subgraph via a matching lower bound. In *Proc. SODA 2001*, pages 417–426. ACM/SIAM, 2001. URL: http://dl.acm.org/citation.cfm?id=365411.365493.

43 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/.

44 Mingui Zhang, Cheng Yi, Bin Liu, and Beichuan Zhang. GreenTE: Power-aware traffic engineering. In *Proc. ICNP 2010*, pages 21–30. IEEE Computer Society, 2010. doi:10.1109/ICNP.2010.5762751.

45 Liang Zhao, Hiroshi Nagamochi, and Toshihide Ibaraki. A linear time 5/3-approximation for the minimum strongly-connected spanning subgraph problem. *Inf. Process. Lett.*, 86(2):63–70, 2003. doi:10.1016/S0020-0190(02)00476-3.

Sparse Induced Subgraphs in P_7-Free Graphs of Bounded Clique Number

Maria Chudnovsky
Princeton University, NJ, USA

Jadwiga Czyżewska
University of Warsaw, Poland

Kacper Kluk
University of Warsaw, Poland

Marcin Pilipczuk
University of Warsaw, Poland

Paweł Rzążewski
Warsaw University of Technology, Poland
University of Warsaw, Poland

Abstract

Many natural computational problems, including e.g. Max Weight Independent Set, Feedback Vertex Set, or Vertex Planarization, can be unified under an umbrella of finding the largest sparse induced subgraph that satisfies some property definable in CMSO_2 logic. It is believed that each problem expressible with this formalism can be solved in polynomial time in graphs that exclude a fixed path as an induced subgraph. This belief is supported by the existence of a quasipolynomial-time algorithm by Gartland, Lokshtanov, Pilipczuk, Pilipczuk, and Rzążewski [STOC 2021], and a recent polynomial-time algorithm for P_6-free graphs by Chudnovsky, McCarty, Pilipczuk, Pilipczuk, and Rzążewski [SODA 2024].

In this work we extend polynomial-time tractability of all such problems to P_7-free graphs of bounded clique number.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases P_t-free graphs, maximum weight induced subgraph, maximum weight independent set

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.20

Related Version *Full Version*: https://arxiv.org/abs/2412.14836

Funding *Maria Chudnovsky*: Supported by NSF Grant DMS-2348219 and by AFOSR grant FA9550-22-1-0083.
Jadwiga Czyżewska: Supported by Polish National Science Centre SONATA BIS-12 grant number 2022/46/E/ST6/00143.
Kacper Kluk: Supported by Polish National Science Centre SONATA BIS-12 grant number 2022/46/E/ST6/00143.
Marcin Pilipczuk: Supported by Polish National Science Centre SONATA BIS-12 grant number 2022/46/E/ST6/00143.
Paweł Rzążewski: Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme grant agreement number 948057.

© Maria Chudnovsky, Jadwiga Czyżewska, Kacper Kluk, Marcin Pilipczuk, and Paweł Rzążewski;
licensed under Creative Commons License CC-BY 4.0

1 Introduction

When studying computationally hard problems, a natural question to investigate is whether they become tractable when input instances are somehow "well-structured". In particular, in algorithmic graph theory, we often study the complexity of certain (hard in general) problems, when restricted to specific graph classes. Significant attention is given to classes that are *hereditary*, i.e., closed under vertex deletion.

Potential maximal cliques. Arguably, the problem that is best studied in this context is Max Weight Independent Set (MWIS) in which we are given a vertex-weighted graph and we ask for a set of pairwise non-adjacent vertices of maximum possible weight. Investigating the complexity of MWIS in restricted graph classes led to discovering numerous new tools and techniques in algorithmic graph theory [24, 21, 18, 16]. One of such general techniques is the framework of *potential maximal cliques* by Bouchitté and Todinca [7, 8]. Intuitively speaking, a potential maximal clique (or PMC for short) in a graph G is a bag of a "reasonable" tree decomposition of G. The key contribution of Bouchitté and Todinca [7, 8] is showing that:

1. the family $\mathcal{F}$ of PMCs in a graph G can be enumerated in time polynomial in $|V(G)|$ and $|\mathcal{F}|$, and
2. given a family $\mathcal{F}$ containing *all* PMCs of G (and possibly some other sets as well), MWIS can be solved in time polynomial in $|V(G)|$ and $|\mathcal{F}|$ by mimicking natural dynamic programming on an appropriate (but unknown) tree decomposition.

Consequently, MWIS admits a polynomial-time algorithm when restricted to any class with polynomially many PMCs. (Here we say that a class $\mathcal{X}$ of graphs has polynomially many PMCs if there exists a polynomial p, such that the number of PMCs in any n-vertex graph in $\mathcal{X}$ is at most $\mathsf{p}(n)$.)

Later this framework was extended by Fomin, Todinca, and Villanger [19] to the problem of finding a large "sparse" (here meaning: of bounded treewidth) induced subgraph satisfying certain CMSO_2 formula ψ. (CMSO_2 stands for *Counting Monadic Second Order* logic, which is a logic where one can use vertex/edge variables, vertex/edge set variables, quantifications over these variables, and standard propositional operands.) Formally, for a given integer d and a fixed CMSO_2 formula ψ with one free set variable, the $(\mathrm{tw} \leq d, \psi)$-MWIS problem (here "MWIS" stands for "max weight induced subgraph") is defined as follows.

$(\mathrm{tw} \leq d, \psi)$-MWIS

Input: A graph G equipped with a weight function $\mathfrak{w}\colon V(G) \to \mathbb{Q}_+$.

Task: Find a pair (Sol, X) such that

- $X \subseteq \mathsf{Sol} \subseteq V(G)$,
- $G[\mathsf{Sol}]$ is of treewidth at most d,
- $G[\mathsf{Sol}] \models \psi(X)$,
- X is of maximum weight subject to the conditions above,

or conclude that no such pair exists.

As shown by Fomin, Todinca, and Villanger [19], every problem expressible in this formalism can be solved in polynomial time on classes of graphs with polynomially many potential maximal cliques.

Note that if $d = 0$ and ψ is a formula satisfied by all sets, then $(\mathrm{tw} \leq d, \psi)$-MWIS is exactly MWIS. It turns out that $(\mathrm{tw} \leq d, \psi)$-MWIS captures several other well known computational problems (see also the discussion in [19, 29]), e.g.;

- Feedback Vertex Set (one of Karp's original 21 NP-complete problems [27]), equivalent by complementation to finding an induced forest of maximum weight,
- Even Cycle Transversal [34, 33, 2], equivalent by complementation to finding an induced graph whose every block is an odd cycle,
- finding a largest weight induced subgraph whose every component is a cycle,
- finding a maximum number of pairwise disjoint and non-adjacent induced cycles.

MWIS in graphs excluding a long induced path. Despite all advantages of the Bouchitté-Todinca framework, its applicability is somehow limited, as there are numerous natural hereditary graph classes that *do not* have polynomially many PMCs. Can we use at least some parts of the framework outside its natural habitat?

A notorious open question in algorithmic graph theory is whether MWIS (and, more generally, $(\mathrm{tw} \leq d, \psi)$-MWIS) can be solved in polynomial time in graphs that exclude a fixed path as an induced subgraph. It is believed that this question has an affirmative answer, which is supported by the existence of a quasipolynomial-time algorithm of Gartland, Lokshtanov, Pilipczuk, Pilipczuk, and Rzążewski [20, 36, 22]. However, if it comes to polynomial-time algorithms, we know much less.

A polynomial-time algorithm for MWIS (as well as for many other problems) in P_4-free graphs (i.e., graphs that exclude a 4-vertex path as an induced subgraph; analogously we define P_t-free graphs for any t), was discovered already in 1980s [14]. In today's terms we would say that these graphs have bounded clique-width and thus polynomial-time algorithms for many natural problems, including $(\mathrm{tw} \leq d, \psi)$-MWIS, follow from a celebrated theorem by Courcelle, Makowsky, and Rotics [15]. However, already the P_5-free case proved to be quite challenging. In 2014, Lokshtanov, Vatshelle, and Villanger [30] showed that Max Weight Independent Set admits a polynomial-time algorithm in P_5-free graphs – by adapting the framework of Bouchitté and Todinca. Let us emphasize that P_5-free graphs might have exponentially many PMCs, so the approach discussed above cannot be applied directly. Instead, Lokshtanov, Vatshelle, and Villanger proved that in polynomial time one can enumerate a family of *some* PMCs, and this family is sufficient to solve MWIS via dynamic programming.

By extending this method (and adding a significant layer of technical complicacy), Grzesik, Klimošová, Pilipczuk, and Pilipczuk [23] managed to show that MWIS is polynomially-solvable in P_6-free graphs.

Some time later, Abrishami, Chudnovsky, Pilipczuk, Rzążewski, and Seymour [1] revisited the P_5-free case and introduced a major twist to the method: they proved that in order to solve MWIS and its generalizations, one does not have to have PMCs exactly, but it is sufficient to enumerate their *containers*: supersets that do not introduce any new vertices of the (unknown) optimum solution. They also proved that containers for *all* PMCs in P_5-free graphs can be enumerated in polynomial time, circumventing the problem of exponentially many PMCs and significantly simplifying the approach of Lokshtanov, Vatshelle, and Villanger [30]. This result yields a polynomial-time algorithm for $(\mathrm{tw} \leq d, \psi)$-MWIS in P_5-free graphs.

By further relaxing the notion of a container (to a *carver*), Chudnovsky, McCarty, Pilipczuk, Pilipczuk, and Rzążewski [13] managed to extend polynomial-time solvability of $(\mathrm{tw} \leq d, \psi)$-MWIS to the class of P_6-free graphs.

Our contribution. In this work we push the boundary of tractability of $(\mathrm{tw} \leq d, \psi)$-MWIS in P_t-free graphs by showing the following result.

▶ **Theorem 1.1.** *For every fixed integers d, k, and a CMSO$_2$ formula ψ, the $(\mathrm{tw} \leq d, \psi)$-MWIS problem can be solved in polynomial time in P_7-free graphs with clique number at most k.*

Quite interestingly, in P_t-free graphs, the boundedness of treewidth is equivalent to the boundedness of degeneracy and to the boundedness of treedepth [4]. Thus, Theorem 1.1 yields a polynomial-time algorithm for problems like:

- Vertex Planarization [26, 35], which asks for a largest induced planar subgraph, or
- for constant k, finding a largest set of vertices inducing a subgraph of maximum degree at most k (see, e.g., [25]).

Let us remark that the idea of investigating P_t-free graphs of bounded clique number was considered before. Pilipczuk and Rzążewski [37] showed that P_6-free graphs of bounded clique number have polynomially many PMCs and thus the framework of Bouchitté and Todinca can be applied here directly. However, this is no longer true for P_7-free graphs (even bipartite) (see the example in the Conclusions in [13]). On the other hand, Brandstädt and Mosca [9] provided a polynomial-time algorithm for MWIS in P_7-free triangle-free graphs. However, they used a rather ad-hoc approach that works well for MWIS, but gives little hope to generalize it to more complicated instances of $(\mathrm{tw} \leq d, \psi)$-MWIS. Our work vastly extends both these results.

2 Technical overview

For standard graph notation used in this work, including the definition of a tree decomposition, we refer to the preliminaries section of the full version in the appendix.

As mentioned, in P_t-free graphs the width parameters treewidth, treedepth, and degeneracy are functionally equivalent [4]. Furthermore, as discussed in [13], the property of having treewidth, treedepth, or degeneracy at most d can be expressed as a CMSO$_2$ formula of size depending on d only. Consequently, if we define problems $(\mathrm{td} \leq d, \psi)$-MWIS and $(\deg \leq d, \psi)$-MWIS analogously as $(\mathrm{tw} \leq d, \psi)$-MWIS, but with respect to treedepth and degeneracy, these three formalisms describe the same family of problems, when restricted to P_t-free graphs.

Treedepth structures. Following [13], the treedepth formalism is the most handy; we henceforth work with $(\mathrm{td} \leq d, \psi)$-MWIS. Furthermore, it is more convenient to work not just with induced subgraphs of bounded treedepth, but with *treedepth-d structures*: induced subgraphs of treedepth at most d with a fixed elimination forest. Let us proceed with formal definitions.

The *rooted forest* $\mathcal{T}$ is a forest in which each component has exactly one distinguished vertex, called a *root*. A path in a rooted forest is called *vertical* if it connects a vertex and any of its ancestors. Given a vertex v, we define its *depth* as the number of vertices on the path connecting v and the root (so the root has depth 1). The *height* of the rooted forest $\mathcal{T}$ is equal to the maximum depth of a vertex of $\mathcal{T}$. By $\mathcal{T}^\alpha$ we denote a set of vertices of $\mathcal{T}$ of depth exactly α. We also write $\mathcal{T}^{>\alpha}$ for $\bigcup_{\alpha'>\alpha} \mathcal{T}^{\alpha'}$.

We say that vertices u and v are *$\mathcal{T}$-comparable* if they can be connected via a vertical path. Otherwise, we say that they are *$\mathcal{T}$-incomparable*. An *elimination forest* of graph G is a rooted forest $\mathcal{T}$ such that $V(\mathcal{T}) = V(G)$ and for each edge $uv \in E(G)$ vertices u and v are $\mathcal{T}$-comparable. The *treedepth* of graph G is the minimum height of any possible elimination forest of graph G.

Given a graph G and an integer d, a *treedepth-d structure* (the notion first proposed in [13]) is a rooted forest $\mathcal{T}$ of height at most d such that $V(\mathcal{T})$ is a subset of $V(G)$ and $\mathcal{T}$ is an elimination forest of the subgraph of G induced by $V(\mathcal{T})$. A treedepth-d structure $\mathcal{T}$ is a *substructure* of a treedepth-d structure $\mathcal{T}'$ if $\mathcal{T}$ is a subgraph of $\mathcal{T}'$ (as a rooted forest) and every root of $\mathcal{T}$ is also a root of $\mathcal{T}'$. A treedepth-d structure $\mathcal{T}$ is called *maximal* if there is no treedepth-d structure $\mathcal{T}'$ such that $\mathcal{T}$ is a substructure of $\mathcal{T}'$ and $\mathcal{T} \neq \mathcal{T}'$. Equivalently, $\mathcal{T}$ is maximal if it cannot be extended by adding a leaf preserving the bound on the height of $\mathcal{T}$. To avoid notational clutter, we sometimes treat $\mathcal{T}$ as set of vertices, for example in expressions $|A \cap \mathcal{T}|$.

Using treedepth structures. Let $\mathcal{T}$ be a treedepth-d structure in G; think of $\mathcal{T}$ as the sought solution to the $(\mathrm{td} \leq d, \psi)$-MWIS problem in question. As proven in [19] (and cast onto the current setting in [13]), there exists a tree decomposition (T, β) of G whose every bag is either contained in $N[v]$ for a leaf v of $\mathcal{T}$, or whose intersection with $\mathcal{T}$ is contained in a single root-to-leaf path of $\mathcal{T}$, excluding the leaf. We call the latter bags *$\mathcal{T}$-avoiding*.

Furthermore, the framework of [19] shows how to solve $(\mathrm{td} \leq d, \psi)$-MWIS given a family $\mathcal{F}$ of subsets of $V(G)$ with the promise that all bags of such tree decomposition (T, β) are in $\mathcal{F}$. In [1], it is shown that it suffices for $\mathcal{F}$ to contain only *containers of bounded defect* for bags in (T, β): we require that there exists a universal constant $\widehat{d}$ such that for every $t \in V(T)$ there exists $A \in \mathcal{F}$ such that the bag $\beta(t)$ is contained in A and A contains at most $\widehat{d}$ elements of $\mathcal{T}$.

Note that if v is a leaf of $\mathcal{T}$, then $N[v]$ may contain only ancestors of v among the vertices of $\mathcal{T}$, so $N[v]$ is an excellent container for any bag contained in $N[v]$. Thus, it suffices to provide containers for $\mathcal{T}$-avoiding bags. This is how [1] solved $(\mathrm{td} \leq d, \psi)$-MWIS in P_5-free graphs; by generalizing the definition of a container to a carver, the same approach is applied to P_6-free graphs in [13].

In fact, for a treedepth-d structure $\mathcal{T}$, the tree decomposition (T, β) of [19] is constructed as follows: it is shown that there exists a minimal chordal completion F of G (i.e., an inclusion-wise minimal set of edges to add to G to make it chordal) whose addition keeps $\mathcal{T}$ a treedepth-d structure, and (T, β) is any clique tree of $G + F$. A *potential maximal clique* (abbrieviated as PMC) is a maximal clique in $G + F$ for *some* minimal chordal completion F of G. Both [1] and [13] actually provide containers/carvers for *every* treedepth-d structure in G and an *arbitrary* choice of a $\mathcal{T}$-avoiding PMC in G. Furthermore, [13] provides an example of a family of P_7-free triangle-free graphs where such a statement is impossible (i.e., any such family of containers/carvers would need to be of exponential size).

However, the algorithmic framework of [19, 1, 13] requires only that the provided family $\mathcal{F}$ contains containers/carvers for all bags of *only one* "good" tree decomposition (T, β), and only for the sought solution $\mathcal{T}$. This should be contrasted with all $\mathcal{T}$-avoiding PMCs, which is in some sense the set of all reasonable $\mathcal{T}$-avoiding bags, and for all treedepth-d structures $\mathcal{T}$. Thus, if we want to tackle P_7-free graphs of bounded clique number, we need to significantly deviate from the path of [1, 13] and either use the power of the choice of (T, β) or the fact that we need to handle only $\mathcal{T}$ being the optimum solution to the problem we are solving.

From PMCs to minimal separators. The assumption of bounded clique number allows us to quickly reduce the case of finding a container for a PMC to finding a container for a minimal separator. For a set $S \subseteq V(G)$, a connected component A of $G - S$ is *full to S* if

$N(A) = S$; a set S is a *minimal separator* if it has at least two full components.[1] A classic fact about PMCs [8] is that for every potential maximal clique Ω and for every connected component D of $G - \Omega$, the set $N(D)$ is a minimal separator. By a classic result of Gyárfás, the class of P_t-free graphs is χ-bounded, in particular, if G is P_t-free and $\omega(G) \leq k$ (where $\omega(G)$ is the number of vertices in a largest clique in G), then $\chi(G) \leq (t-1)^{k-1}$. This, combined with a VC-dimension argument based on the classic result of Ding, Seymour, and Winkler [17], shows the following:

▶ **Lemma 2.1.** *For every t and k there exists c such that for every P_t-free graph G with $\omega(G) \leq k$ and any PMC Ω of G one of the two following conditions holds:*
1. *there exists $v \in \Omega$ such that $\Omega = N[v]$, or*
2. *there exists a family $\mathcal{D} \subseteq \mathrm{cc}(G - \Omega)$ of size at most c such that $\Omega = \bigcup_{D \in \mathcal{D}} N(D)$.*

That is, except for a simple case $N[v] = \Omega$ for some $v \in \Omega$, the PMC Ω is a union of a constant number of minimal separators. Hence, it suffices to generate a family $\mathcal{F}'$ of containers for $\mathcal{T}$-avoiding minimal separators (which are defined analogously to $\mathcal{T}$-avoiding bags) and take $\mathcal{F}$ to be the family of unions of all tuples of at most c elements of $\mathcal{F}'$.

Capturing minimal separators. Let G be a P_7-free graph with $\omega(G) \leq k$. We start as in [1, 13]: let $\mathcal{T}$ be an arbitrary treedepth-d structure in G and let S be a minimal separator in G with full components A and B such that S is $\mathcal{T}$-avoiding. We want to design a polynomial-time algorithm that outputs a family $\mathcal{F}'$ of subsets of $V(G)$ that contains a container for S. The algorithm naturally does not know $\mathcal{T}$ nor S; the convenient and natural way of describing the algorithm as performing some nondeterministic guesses about $\mathcal{T}$ and S, with the goal of outputting a container for S in the end. We succeed if the number of subcases coming from the guesses is polynomial in the size of G and the family $\mathcal{F}'$ consists of all containers generated by all possible runs of the algorithm.

We almost succeed with this quest. That is, we are able to guess an "almost container" K for S: K contains a constant number of vertices of $\mathcal{T}$ and we identified a set $\mathcal{D}$ of *tricky* connected components of $G - K$ that are contained in $A \cup B \cup S$; all other connected components of $G - K$ are contained in A, in B, or in some other connected component of $G - S$. Every tricky component $D \in \mathcal{D}$ is somewhat simpler: we identify a subset $\emptyset \neq L(D) \subsetneq D$ such that if $\mathcal{C}(D)$ is the family of all connected components of $G[L(D)]$ and of $G[D \setminus L(D)]$, then every $C \in \mathcal{C}(D)$ is a module of $G[D]$. (A set $A \subseteq V(G)$ is a module in G if every vertex $v \in V(G) \setminus A$ is either complete to A or anti-complete to A.)

Observe now that every connected component $C \in \mathcal{C}(D)$ satisfies $\omega(G[C]) < \omega(G) \leq k$, as $D \setminus C$ contains a vertex complete to C. Hence, we can recurse on C, understanding it fully (more precisely, finding optimum partial solutions; the definition of this "partial" requires a lot of CMSO_2 mumbling).

Furthermore, we observe that the quotient graph $G[D]/\mathcal{C}(D)$ is bipartite. Thus, if we can understand how to solve $(\mathrm{td} \leq d, \psi)$-MWIS on P_7-free *bipartite* graphs, then we should be done: the partial solutions from components $C \in \mathcal{C}(D)$, combined with the understanding of the P_7-free bipartite graph $G[D]/\mathcal{C}(D)$ should give all partial solutions of $G[D]$ for every $D \in \mathcal{D}$. This, in turn, should give an understanding on how $(\mathrm{td} \leq d, \psi)$-MWIS behaves on $K \cup \bigcup \mathcal{D}$. As $S \subseteq K \cup \bigcup \mathcal{D}$, this should suffice to solve $(\mathrm{td} \leq d, \psi)$-MWIS on G using $(K, \mathcal{D}, L)$ to play the role of the container for S.

[1] Note the following equivalent definition: We say that a set of vertices S is a (u,v)- *separator* if u and v belong to distinct components in $G - S$. A (u,v)-separator S is called *minimal* if no proper subset of S is a (u,v)-separator. A set of vertices S is called a *minimal separator* of graph G if there exists a pair of vertices u and v such that S is a minimal (u,v)-separator.

The bipartite case. Explaining properly all "should"s from the previous paragraph is quite involved and tedious. In this overview, let us focus on the more interesting case of solving $(\mathrm{td} \leq d, \psi)$-MWIS in P_7-free bipartite graphs. Here, we actually prove the following result.

▶ **Theorem 2.2.** *There exists an algorithm that, given a P_7-free bipartite graph G and an integer $d > 0$, runs in time $n^{2^{2^{\mathcal{O}(d^3)}}}$ and computes a family $\mathcal{C}$ of subsets of $V(G)$ with the following guarantee: for every $\mathsf{Sol} \subseteq V(G)$ such that $G[\mathsf{Sol}]$ is of treedepth at most d, there exists a tree decomposition (T, β) of G such that for every $t \in V(T)$ we have $\beta(t) \in \mathcal{C}$ and $|\beta(t) \cap \mathsf{Sol}| = 2^{2^{\mathcal{O}(d^3)}}$.*

We remark that Theorem 2.2 is not needed if one just wants to solve the MWIS problem, as this problem can be solved in general bipartite graphs using matching or flow techniques.

On the other hand, our work identified the bipartite case as an interesting subcase in exploring tractability of $(\mathrm{td} \leq d, \psi)$-MWIS in P_t-free graphs. To state a precise problem for future work, we propose polynomial-time tractability of Feedback Vertex Set in P_t-free bipartite graphs. Meanwhile, Theorem 2.2 and its proof in Section 3 can be of independent interest.

The proof of Theorem 2.2 starts from the work of Kloks, Liu, and Poon [28], who showed that *chordal bipartite* graphs have polynomial number of PMCs, and thus $(\mathrm{td} \leq d, \psi)$-MWIS problem is solvable in this graph class by the direct application of the PMC framework of Bouchitté and Todinca. (A graph G is *chordal bipartite* if it is bipartite and does not contain induced cycles longer than 4.) A P_7-free bipartite graph is *almost* chordal bipartite: it can contain six-vertex cycles.

We would like to add some edges to the input graph G so that it becomes chordal bipartite. Let $C = c_1 - c_2 - \ldots - c_6 - c_1$ be an induced six-vertex cycle in G; we would like to add the edge c_1c_4 to keep G bipartite but break C. We show that, if one chooses C carefully, one can do it so that G remains P_7-free. However, such an addition may break the sought solution $\mathcal{T}$: if $c_1, c_4 \in V(\mathcal{T})$ but are incomparable in the elimination forest $\mathcal{T}$, then $\mathcal{T}$ is no longer a treedepth-d structure in $G + \{c_1c_4\}$.

We remedy this by a thorough investigation of the structure of the neighborhood of a six-vertex cycle in a P_7-free graph, loosely inspired by [5]. On a high level, we show that there is a branching process with a polynomial number of outcomes that, in some sense "correctly" completes G to a chordal bipartite graph, proving Theorem 2.2.

In summary, the proof of Theorem 1.1 consists of three ingredients. First, we show the guesswork that leads to a polynomial number of candidate "almost containers" $(K, \mathcal{D}, L)$ for a minimal separator S in the input graph G. Second, we focus on bipartite graphs and prove Theorem 2.2. Finally, we show how these two tools combine with the dynamic programming framework of [19, 1, 13].

In this extended abstract, we present a more detailed overview of the proof of one part, namely Theorem 2.2. The full version of the work can be found in the appendix.

3 P_7-free bipartite graphs: Overview of the proof of Theorem 2.2

In this section we focus on P_7-free bipartite graphs. We prove the following theorem.

▶ **Theorem 2.2.** *There exists an algorithm that, given a P_7-free bipartite graph G and an integer $d > 0$, runs in time $n^{2^{2^{\mathcal{O}(d^3)}}}$ and computes a family $\mathcal{C}$ of subsets of $V(G)$ with the following guarantee: for every $\mathsf{Sol} \subseteq V(G)$ such that $G[\mathsf{Sol}]$ is of treedepth at most d, there exists a tree decomposition (T, β) of G such that for every $t \in V(T)$ we have $\beta(t) \in \mathcal{C}$ and $|\beta(t) \cap \mathsf{Sol}| = 2^{2^{\mathcal{O}(d^3)}}$.*

The general approach is to reduce to the case of *chordal bipartite graphs*. Recall that a bipartite graph is *chordal bipartite* if its every cycle of length longer than 4 has a chord (i.e., the only induced cycles are of length 4).

The class of MWIS problems is tractable on chordal bipartite graphs thanks to the following result of Kloks, Liu, and Poon (and the general algorithm of Fomin, Todinca, and Villanger [19]).

▶ **Theorem 3.1** (Corollary 2 of [28]). *A chordal bipartite graph on n vertices and m edges has $\mathcal{O}(n+m)$ minimal separators.*

Observe that P_7-free bipartite graphs are "almost" chordal bipartite: they only additionally allow C_6 as an induced subgraph. Our approach is to add edges to the input graph so that it becomes chordal bipartite, without destroying the sought solution. To this end, the following folklore characterization will be handy. (We use ♠ to mark statements whose proofs are omitted in this extended abstract due to space constraints.)

▶ **Lemma 3.2** (folklore, ♠). *Let G be a bipartite graph with bipartition V_1, V_2. Then, G is chordal bipartite if and only if for every minimal separator S of G, $S \cap V_1$ is complete to $S \cap V_2$ (i.e., the separator induces a complete bipartite graph, also called a* biclique*).*

Lemma 3.2 motivates the following process of completing a P_7-free bipartite graph G into a chordal bipartite graph: while G contains a minimal separator S that violates the statement of Lemma 3.2, complete $G[S]$ into a biclique. In Section 3.1 we analyse this process and show that it is well-behaved on P_7-free graphs: it does not lead outside the class of P_7-free bipartite graphs.

3.1 Completing to a chordal bipartite graph

Throughout the rest of this section we assume that the input graph G has a fixed bipartition into sets V_1, V_2 (this is an ordered partition). We remark that we will often look at certain induced subgraphs of G that are not necessarily connected, but a vertex never changes its side of the bipartition.

Lemma 3.2 motivates the following definition. Let G be a bipartite graph with bipartition V_1, V_2, and let S be a minimal separator of G. We say that S *induces a biclique* if $S \cap V_1$ is complete to $S \cap V_2$. The operation of *completing S into a biclique* turns G into a graph $G + F := (V(G), E(G) \cup F)$, where $F = \{uv \mid u \in S \cap V_1, v \in S \cap V_2, uv \notin E(G)\}$.

The next two lemmata (proved by a tedious case analysis) are pivotal to our completion process.

▶ **Lemma 3.3** (♠). *Let G be a P_7-free bipartite graph. Let S be a minimal separator. Let $G + F$ be the result of completing S into a biclique. Then $G + F$ is also P_7-free.*

▶ **Lemma 3.4** (♠). *Let G be a P_7-free bipartite graph. Let S be a minimal separator in G. Let $G + F$ be the result of completing S into a biclique. Let C be an induced C_6 in $G + F$. Then, C is also an induced C_6 in G.*

Let G be a P_7-free bipartite graph with fixed bipartition V_1, V_2, and let $\mathcal{T}$ be a treedepth-d structure in G. A tuple (C, x, y) is a *bad C_6* (with respect to $\mathcal{T}$) in G if C is an induced six-vertex cycle in G, and $x \in V_1$, $y \in V_2$ are two vertices that are at the same time (a) opposite vertices of C, and (b) incomparable vertices of $\mathcal{T}$. That is, if one adds the edge xy to G (which can be done without violating the biparteness of G), then one breaks the treedepth-d structure $\mathcal{T}$, i.e., $\mathcal{T}$ is not a treedepth-d structure in $G + xy$.

We have the following simple corollary of Lemma 3.4.

▶ **Lemma 3.5.** *Let G be a P_7-free bipartite graph, let $\mathcal{T}$ be a treedepth-d structure in G, let S be a minimal separator in G, and let $G+F$ be a result of completing S into a biclique. Assume that there is no bad C_6 in G with respect to $\mathcal{T}$. Then, $\mathcal{T}$ is a treedepth-d structure in $G+F$ and, furthermore, there is no bad C_6 in $G+F$ with respect to $\mathcal{T}$.*

Proof. By contradiction, suppose that $E(C) \cap F \neq \emptyset$. As S in a biclique in $G+F$, $V(C) \cap S$ contains either two or three consecutive vertices of C. Thus, $P := C \setminus S$ is a path and belongs to exactly one component of $G - S$. Without loss of generality, we can assume that P does not lie in B, i.e., $C \cap B = \emptyset$.

Let $x, z \in V(C) \cap S$ be the two vertices of C adjacent on C to the vertices of P. Note that $xz \notin E(G)$: either $|V(C) \cap S| = 3$ and x, z are on the same biparteness side of G or $|V(C) \cap S| = 2$ and then $xz \in F$.

Let R be the shortest path from x to z via B. Then, $R \cup P$ induce a hole C' in G. Since R is of length at least 2, C' is not shorter than C. Since G is P_7-free, C' is a six-vertex hole. This can only happen if R is of length 2 and $V(C) \cap S = \{x, y, z\}$ for some $y \in S$. Then, $C = x - y - z - r - q - p - x$, where p, q, r lie in a single component of $G - S$ different than B. Without loss of generality, we can assume that $yz \in F$. Then we can find a shortest path R', which connects y and z via B; it is of length at least 3. Then the path $R - r - q - p$ is an induced path in G on at least 7 vertices, a contradiction. ◀

We conclude this section with the following enumeration.

▶ **Lemma 3.6.** *There exists an algorithm that, given a P_7-free bipartite graph G and an integer d, runs in polynomial time and returns a family $\mathcal{C}$ of subsets of $V(G)$ of size $\mathcal{O}(|V(G)|^5)$ with the following property: for every treedepth-d structure $\mathcal{T}$ in G that admits no bad C_6, there exists a tree decomposition (T, β) of G such that for every $t \in V(T)$, the set $\beta(t)$ belongs to $\mathcal{C}$ and $\beta(t)$ contains at most d elements of $\mathcal{T}$.*

Proof. Consider the following process. Start with $\widehat{G} := G$. While $\widehat{G}$ is not chordal bipartite, find an induced six-vertex cycle C in G, fix two opposite vertices x and y in C, find a minimal separator S in G that contains x and y (note that any minimal separator separating the two components of $C - \{x, y\}$ would do), complete S into a biclique obtaining $\widehat{G} + F$, and set $\widehat{G} := \widehat{G} + F$.

Lemma 3.3 ensures that $\widehat{G}$ stays P_7-free bipartite. Lemma 3.5 ensures that every treedepth-d structure $\mathcal{T}$ in G that admits no bad C_6 remains so in $\widehat{G}$.

By Theorem 3.1, $\widehat{G}$ has $\mathcal{O}(|V(G)|^2)$ minimal separators. Using [7, 8], this allows us to bound the number of potential maximal cliques of $\widehat{G}$ by $\mathcal{O}(|V(G)|^5)$ and enumerate them in a family $\mathcal{C}$.

As shown in [13], there exists a minimal chordal completion F of $\widehat{G}$ such that for every maximal clique Ω of $\widehat{G} + F$, the set $\Omega \cap \mathcal{T}$ is contained in a single leaf-to-root path in $\mathcal{T}$ and thus is of size at most d. Since all these maximal cliques are enumerated in $\mathcal{C}$, any clique tree of $\widehat{G} + F$ serves as the promised tree decomposition (T, β). ◀

3.2 Cleaning

In this section we define a branching step that cleans a specific part of the graph. The step will be general enough to be applicable in many contexts.

Let G be a P_7-free bipartite graph. We say that a triple (A, B, C) of pairwise disjoint vertex sets of G is $\bigvee$*-free* if there are no two anticomplete P_3s of the form $A - B - C$. Note that such a configuration naturally appears in a P_7-free bipartite graph if $A \cup C$ is in one side on the bipartition, B is in the other side, and there is an additional vertex v in the same side as B such that $A \subseteq N(v)$ but $C \cap N(v) = \emptyset$. In our case, $\bigvee$-free sets appear naturally in the following context.

▶ **Lemma 3.7.** *Let G be a P_7-free bipartite graph with a fixed bipartition V_1, V_2. Let A and C be two disjoint subsets contained in V_i, for some $i \in \{1,2\}$, and let $B \subseteq V_{3-i}$. Furthermore, assume that there exists a vertex $v \in V_{3-i} \setminus B$ such that $A \subseteq N(v)$ but $C \cap N(v) = \emptyset$. Then, (A,B,C) is Ψ-free.*

Proof. Any two anticomplete P_3s of the form $A - B - C$, together with v, induce a P_7 in G. ◀

Recall that in our setting, we have some unknown treedepth-d structure $\mathcal{T}$ in G and we are building a container for it. That is, we are happy with inserting any number of vertices of G into the container, as long as we are guaranteed that we insert only a constant number of vertices of $\mathcal{T}$ along the way. In the case of a Ψ-free triple (A,B,C), we would like to simplify this part of G by filtering out vertices of B that have neighbors both in A and in C. To achieve this goal, we will guess a set $X \subseteq A \cup B \cup C$ that on one hand contains (in one of the branches) at most a constant number of vertices of the fixed unknown $\mathcal{T}$, and on the other hand satisfies the following: no $b \in B \setminus X$ has a neighbor both in $A \setminus X$ and in $C \setminus X$.

To this end, we will rely on the following simple yet powerful observation. Observe that if (A,B,C) is Ψ-free and both $A \cup C$ and B are independent sets (which happen, in particular, if $A \cup C$ is on one side of the bipartition and B is on the other side of the bipartition), then, for every distinct $b_1, b_2 \in B$ either $N(b_1) \cap A$ and $N(b_2) \cap A$ are comparable by inclusion or $N(b_1) \cap C$ and $N(b_2) \cap C$ are comparable by inclusion. This allows to use the following lemma on subsets of B, with the orders $\leq_1$ and $\leq_2$ being inclusion of the neighborhoods in A and C, respectively.

▶ **Lemma 3.8** (see Lemma 4.1 in [23])**.** *Let $(X, \leq_1)$ and $(X, \leq_2)$ be two partial orders on the set X such that any pair of elements of X is comparable in $\leq_1$ or in $\leq_2$. Then, there exists an element v such that for any element $x \in X$ we have $v \leq_1 x$ or $v \leq_2 x$.*

This observation is the engine of the following lemma that formalizes our goal of filtering out vertices of B that have neighbors both in A and in C.

▶ **Lemma 3.9.** *Fix an integer d. Let G be a graph, and let (A,B,C) be a Ψ-free triple in G with both $A \cup C$ and B being independent set. Then one can in polynomial time enumerate a family $\mathcal{F}$ of subsets of $A \cup B \cup C$ of size at most $n^{4(d+1)^2}$ with the following guarantee:*

- *For every $X \in \mathcal{F}$, there is no $b \in B \setminus X$ with both $N(b) \cap (A \setminus X)$ and $N(b) \cap (C \setminus X)$ nonempty.*
- *For every treedepth-d structure $\mathcal{T}$ in G, there exists $X \in \mathcal{F}$ such that $|X \cap \mathcal{T}| \leq d^2(d+1)$.*

Proof. Fix a treedepth-d structure $\mathcal{T}$ in G. We will describe the process of enumerating the elements of $\mathcal{F}$ as a branching algorithm, guessing some properties of $\mathcal{T}$. The number of leaves of the branching will be polynomial in the size of G and in each leaf of the branching we will output one set X that satisfies the second property for every $\mathcal{T}$ that agrees with the guesses made in this leaf.

For every $b \in B$, let $\iota_A(b)$ be the maximum integer $1 \leq \alpha \leq d$ such that $N(b) \cap A \cap \mathcal{T}^\alpha$ contains at least two vertices; $\iota_A(b) = 0$ if such a α does not exist. Similarly define $\iota_C(b)$ with respect to $N(b) \cap C \cap \mathcal{T}^\alpha$. For $0 \leq \alpha, \beta \leq d$, let $B_{\alpha,\beta} = \{b \in B \mid \iota_A(b) = \alpha \wedge \iota_C(b) = \beta\}$. Note that sets $B_{\alpha,\beta}$ form a partition of B.

Initialize $X = \emptyset$. For every $0 \leq \alpha, \beta \leq d$, we we will guess some set of vertices and include them into X. We will argue that there will be a branch where the vertices guessed for α, β ensure that every vertex from $B_{\alpha,\beta} \setminus X$ satisfies the first statement of the lemma. Thus, for (at least) one of sets X generated in the process, the statement will hold for every vertex in $B \setminus X$.

Consider fixed $0 \leq \alpha, \beta \leq d$. If $B_{\alpha,\beta}$ is empty, there is nothing to do, so assume otherwise. Define the following two orders $\leq_1$ and $\leq_2$ on $B_{\alpha,\beta}$:

$$\begin{aligned}
b \leq_1 b' &\iff N(b) \cap A \supseteq N(b') \cap A, && \text{if } \alpha = 0,\\
b \leq_1 b' &\iff N(b) \cap A \cap \mathcal{T}^\alpha \subseteq N(b') \cap A \cap \mathcal{T}^\alpha && \text{if } \alpha > 0,\\
b \leq_2 b' &\iff N(b) \cap C \supseteq N(b') \cap C, && \text{if } \beta = 0,\\
b \leq_2 b' &\iff N(b) \cap C \cap \mathcal{T}^\beta \subseteq N(b') \cap C \cap \mathcal{T}^\beta && \text{if } \beta > 0.
\end{aligned}$$

Apply Lemma 3.8 to $B_{\alpha,\beta}$ with $\leq_1$ and $\leq_2$, obtaining a vertex $b^*_{\alpha,\beta}$.

If $\alpha = 0$, guess $b^*_{\alpha,\beta}$ and add $N(b^*_{\alpha,\beta}) \cap A$ to X. This adds at most d vertices of $\mathcal{T}$ to X, while adds to X all vertices of $N(b) \cap A$ such that $b \in B_{\alpha,\beta}$ and $b^*_{\alpha,\beta} \leq_1 b$.

If $\alpha > 0$, guess two elements $a^1_{\alpha,\beta}, a^2_{\alpha,\beta} \in N(b^*_{\alpha,\beta}) \cap A \cap \mathcal{T}^\alpha$ and add $N(a^1_{\alpha,\beta}) \cap N(a^2_{\alpha,\beta}) \cap B$ to X. This adds at most $(\alpha - 1)$ vertices of $\mathcal{T}$ to X, while adds to X every $b \in B_{\alpha,\beta}$ such that $b^*_{\alpha,\beta} \leq_1 b$.

Perform symmetrical operation for β and C: If $\beta = 0$, guess $b^*_{\alpha,\beta}$ and add $N(b^*_{\alpha,\beta}) \cap C$ to X. This adds at most d vertices of $\mathcal{T}$ to X, while adds to X all vertices of $N(b) \cap C$ such that $b \in B_{\alpha,\beta}$ and $b^*_{\alpha,\beta} \leq_2 b$. If $\beta > 0$, guess two elements $c^1_{\alpha,\beta}, c^2_{\alpha,\beta} \in N(b^*_{\alpha,\beta}) \cap C \cap \mathcal{T}^\beta$ and add $N(c^1_{\alpha,\beta}) \cap N(c^2_{\alpha,\beta}) \cap B$ to X. This adds at most $(\beta - 1)$ vertices of $\mathcal{T}$ to X, while adds to X every $b \in B_{\alpha,\beta}$ such that $b^*_{\alpha,\beta} \leq_2 b$.

Finally, add the resulting set X to $\mathcal{F}$ if it satisfies the first bullet point of the statement.

We have already argued that in the branch when all guesses concerning $\mathcal{T}$ are correct, the first bullet point of the statement is satisfied. Furthermore, the size of $\mathcal{T} \cap X$ is bounded by

$$\sum_{\alpha=1}^{d} \sum_{\beta=1}^{d} (\alpha + \beta) = 2d \, \frac{d(d+1)}{2} = d^2(d+1).$$

Finally, observe that the number of branches is bounded by $n^{4(d+1)^2}$ as for every $0 \leq \alpha, \beta \leq d$ we guess at most four vertices of G. ◀

Thanks to Lemma 3.7, Lemma 3.9 is applicable in the following setting.

▶ **Definition 3.10.** *Let G be a P_7-free bipartite graph with a fixed bipartition V_1, V_2 and let $Z \subseteq V(G)$. The* neighborhood partition with respect to Z *is a partition $\mathcal{Z}$ of $V(G) \setminus Z$ into sets depending on (1) their side in the bipartition, and (2) their neighborhood in Z. More precisely,*

$$\mathcal{Z} = \{A_{i,Y} \mid i \in \{1,2\}, Y \subseteq Z \cap V_{3-i}\}, \quad \text{where} \quad A_{i,Y} = \{v \in (V(G) \setminus Z) \cap V_i \mid N(v) \cap Z = Y\}.$$

▶ **Lemma 3.11.** *Let G be a P_7-free bipartite graph with a fixed bipartition V_1, V_2, let $Z \subseteq V(G)$, and let $\mathcal{Z}$ be the neighborhood partition with respect to Z. Then, for every distinct $A, B, C \in \mathcal{Z}$, the triple (A, B, C) is Ψ-free.*

Proof. There is no P_3 of the form $A - B - C$ unless A and C are on one side of the bipartition, and B is on the other side of the bipartition. Then, by the definition of $\mathcal{Z}$, the vertices of A and the vertices of C differ in their neighborhood in Z: there exists $v \in Z$ such that either $N(v) \cap (A \cup C) = A$ or $N(v) \cap (A \cup C) = C$. Then, the claim follows from Lemma 3.7 and the symmetry of A, C. ◀

We summarize this section with the following cleaning step.

▶ **Lemma 3.12.** *Let G be a P_7-free bipartite graph with a fixed bipartition V_1, V_2, let $Z \subseteq V(G)$, and let $\mathcal{Z}$ be the neighborhood partition with respect to Z. Then one can in time $|V(G)|^{2^{3|Z|} \cdot \mathcal{O}(d^2)}$ enumerate a family $\mathcal{F}$ of at most $n^{4(d+1)^2 2^{3|Z|}}$ subsets of $V(G) \setminus Z$ with the following properties:*

- *For every $X \in \mathcal{F}$, for every $v \in V(G) \setminus X$, the elements of $N(v) \setminus (X \cup Z)$ are contained in a single set of $\mathcal{Z}$.*
- *For every treedepth-d structure $\mathcal{T}$ in G, there exists $X \in \mathcal{F}$ such that $|X \cap \mathcal{T}| \leq 2^{3|Z|} \cdot d^2(d+1)$.*

Proof. Observe that $|\mathcal{Z}| \leq 2^{|Z|}+1$. Let $\mathcal{Z}^3$ be the family of all triples (A, B, C) where A, B, C are distinct elements of $\mathcal{Z}$. For every triple $(A, B, C) \in \mathcal{Z}^3$, we apply Lemma 3.9 obtaining a family $\mathcal{F}_{(A,B,C)}$; Lemma 3.11 asserts that the assumptions are satisfied. For every triple $(X_{(A,B,C)})_{(A,B,C)\in\mathcal{Z}^3}$ $\in \prod_{(A,B,C)\in\mathcal{Z}^3} \mathcal{F}_{(A,B,C)}$ insert $\bigcup_{(A,B,C)\in\mathcal{Z}^3} X_{(A,B,C)}$ into $\mathcal{F}$. The promised guarantees and size bounds are immediate from Lemma 3.9 and the bound $|\mathcal{Z}| \leq 2^{|Z|+1}$, which implies $|\mathcal{Z}^3| \leq (2^{|Z|}+1) \cdot 2^{|Z|} \cdot (2^{|Z|}-1) \leq 2^{3|Z|}$. ◀

3.3 Branching on a bad C_6s

The tools developed in Section 3.1 allow us to solve $(\mathrm{td} \leq d, \phi)$-MWIS on G, assuming that there is no bad C_6 w.r.t. the solution treedepth-d structure $\mathcal{T}$. We now investigate properties of bad C_6s and how to branch on them.

We will often denote an induced C_6 as $C = c_1 - c_2 - \ldots - c_6 - c_1$. Then (c_1, c_4), (c_2, c_5), (c_3, c_6) are pairs of the opposite vertices in C. We implicitly assume that the indices behave cyclically, i.e., $c_7 = c_1$ etc. We will need the following observation implicit in [6].

▶ **Lemma 3.13 (♠).** *Let G be a P_7-free bipartite graph, let $C = c_1 - c_2 - \ldots - c_6 - c_1$ be an induced C_6 in G, and let D be a connected component of $G - N[V(C)]$ that contains at least two vertices. Then, for every $v \in N(D)$, the set $N(v) \cap V(C)$ equals to either $\{c_1, c_3, c_5\}$ or $\{c_2, c_4, c_6\}$.*

For an induced six-vertex-cycle $C = c_1 - c_2 - \ldots - c_6 - c_1$, denote

$$S_1^C = \{v \in N(V(C)) \mid N(v) \cap V(C) = \{c_1, c_3, c_5\}\},$$
$$S_2^C = \{v \in N(V(C)) \mid N(v) \cap V(C) = \{c_2, c_4, c_6\}\}.$$

Furthermore, let MR_C be the union of vertex sets of all connected components of $G - N[V(C)]$ that contain at least 2 vertices. (MR stands for the "main remainder".) With this notation, Lemma 3.13 states that $N(\mathsf{MR}_C) \subseteq S_1^C \cup S_2^C$.

Greatly simplifying, our algorithm will guess a bad C_6 (C, x, y), resolve $N[V(C)]$ using cleaning, and recurse on connected components of MR_C. To restrict the space of possible recursive calls, we need the following observation.

▶ **Lemma 3.14.** *Let G be a P_7-free bipartite graph with a fixed bipartition V_1, V_2, and let $\mathcal{B}$ be a family of pairwise disjoint and anticomplete subsets of $V(G)$ such that for every $B \in \mathcal{B}$, $G[B]$ is connected and $|B| > 1$. Let D be a connected component of $G - \bigcup_{B\in\mathcal{B}} N[B]$, let C be the connected component of G that contains D. Then, for every $i \in \{1, 2\}$, either $N(D) \cap V_i = \emptyset$ or there exists a single set $B_i \in \mathcal{B}$ such that $N(D) \cap V_i \subseteq N(B_i)$. Consequently, there exists a subfamily $\mathcal{B}' \subseteq \mathcal{B}$ of size at most 2 such that $N(D) \subseteq \bigcup_{B\in\mathcal{B}'} N[B]$ (i.e., D is a connected component of $G - \bigcup_{B\in\mathcal{B}'} N[B]$).*

Proof. Let $\mathcal{B}_i \subseteq \mathcal{B}$ be an inclusion-wise minimal set such that $\bigcup_{B\in\mathcal{B}_i} N(B) \supseteq N(D) \cap V_i$. Assume $|\mathcal{B}_i| > 1$; let $B, B' \in \mathcal{B}_i$ be distinct. By minimality, there exists $v, v' \in N(D) \cap V_i$ such that $v \in N(B) \setminus N(B')$ and $v' \in N(B') \setminus N(B)$. Since $|B|, |B'| > 1$, there exists an induced P_3 of the form $v - B - B$ and an induced P_3 of the form $v' - B' - B'$. Let Q be a shortest path from v to v' via D. Then, there exists an induced path in G of the form $B - B - v - Q - v' - B' - B'$ and this path has at least 7 vertices, a contradiction. ◀

To see how Lemma 3.14 is useful, consider a hypothetical recursive branching algorithm that guesses a bad C_6 (C, x, y) and recurses on connected components of MR_C. Assume that some recursive call is invoked on an induced subgraph $G[A]$ and the C_6s in the stack of the recursion are $C^1, C^2, \ldots, C^p$. Then, $\mathcal{B} = \{V(C^i) \mid 1 \leq i \leq p\}$ satisfies the prerequisites of Lemma 3.14 and $G[A]$ is a connected component of $G - \bigcup_{B\in\mathcal{B}} N[B]$. Lemma 3.14 asserts that there are at most two C_6s among $C^1, C^2, \ldots, C^p$, say C^a and C^b, such that $G[A]$ is a connected component of $G - (N[V(C^a)] \cup N[V(C^b)])$. Since $|V(C^a) \cup V(C^b)| = 12$, there are $\mathcal{O}(|V(G)|^{13})$ options for the set A, and we have a polynomial bound on the number of different possible recursive calls.

However, cleaning $N[V(C)]$ for a bad C_6 (C, x, y) is far from trivial, and we help ourselves with a careful choice of (C, x, y). Informally speaking, we choose the one in which the depths of x and y in $\mathcal{T}$ are maximum possible. This maximality allows us to argue that some other C_6s found in the vinicity of C are not bad, as their vertices of $\mathcal{T}$ are deeper.

Further complications arise from the fact that either S_1^C or S_2^C can contain an arbitrary number of vertices of $\mathcal{T}$ (but we can prove that one of these sets contains at most d vertices of $\mathcal{T}$). In this case, the following observation helps us get enough structure to perform branching.

▶ **Lemma 3.15.** *Let G be a P_7-free bipartite graph with a fixed bipartition V_1, V_2, let $Z \subseteq V(G)$ be connected, and let D, D' be two connected components of $G - N[Z]$ that are of size at least 2 each. Then, for every $i \in \{1, 2\}$, the sets $N(D) \cap V_i$ and $N(D') \cap V_i$ are comparable by inclusion.*

Proof. Assume the contrary, let $x \in (N(D) \setminus N(D')) \cap V_i$ and $x' \in (N(D') \setminus N(D)) \cap V_i$. By the existence of x and x', we observe that Z, x, x', D, D' lie all in the same connected component of G and $x, x' \in N(Z)$. Let Q be the shortest path from x to x' via Z. Since $|D|, |D'| > 1$, there exists an induced P_3 of the form $x - D - D$ and an induced P_3 of the form $x' - D' - D'$. But then there exists an induced path of the form $D - D - x - Q - x' - D' - D'$, which has at least 7 vertices, a contradiction. ◀

The most frequent usage of Lemma 3.15 will be when $Z = V(C)$ for some bad C_6 (C, x, y). Then, D, D' are connected components of MR_C and Lemma 3.15 asserts that for every $i \in \{1, 2\}$, the neighborhoods $N(D) \cap S_i^C$ and $N(D') \cap S_i^C$ are comparable by inclusion.

This concludes the main insights into the proof of Theorem 2.2. Full proof can be found in the full version of the paper in the appendix.

4 Conclusion

Our work suggests some directions for future research. Of course the most ambitious goal is to show a polynomial-time algorithm for $(\mathrm{tw} \leq d, \psi)$-MWIS in P_t-free graphs, for all fixed t, with the first open case for $t = 7$ (with no further assumptions on the graph). However, there are several intermediate goals that also seem interesting. For example, we think that

the idea of considering graphs of bounded clique number is quite promising. Not only it allows us to use some strong structural tools like χ-boundedness or VC-dimension, but also to measure the progress of an algorithm by decreasing clique number (see also [12, 11]).

▶ **Problem 4.1** ([12, Problem 9.2]). *Show that for every fixed t and k, MWIS is polynomial-time solvable in P_t-free graphs with clique number at most k.*

As illustrated in Section 3, assuming additionally that a graph is bipartite might lead to an easier, but still interesting problem. Of course in this setting asking for an algorithm for MWIS makes little sense, but already the next case is far from trivial.

▶ **Problem 4.2.** *Show that for every fixed t, given a vertex-weighted bipartite P_t-free graph, in polynomial-time we can find an induced forest of maximum possible weight.*

Let us point out that in case $t = 7$ one can use an approach alternative to our Theorem 2.2. Indeed, it follows from the work of Lozin and Zamaraev [32] that P_7-free bipartite graphs have bounded mim-width, which allows us to solve maximum induced forest in polynomial time [3]. However, already P_8-free bipartite graphs have unbounded mim-width [10].

A different subclass of P_t-free graphs are $(P_t, K_{1,k})$-free graphs (i.e., excluding additionally a star with k leaves as an induced subgraph), studied by Lozin and Rautenbach [31], who show that Maximum Weight Independent Set is polynomial-time solvable in these graph classes. Does this result generalize to $(\mathrm{tw} \leq d, \psi)$-MWIS?

Finally, let us point out that Abrishami, Chudnovsky, Pilipczuk, Rzążewski, and Seymour [1] did not only provide an algorithm for $(\mathrm{tw} \leq d, \psi)$-MWIS in P_5-free graphs, but they actually considered a richer class of graphs. In particular, their algorithm works for $C_{>4}$-free graphs, i.e., graph with no induced cycles of length more than 4 (analogously we define $C_{>t}$-free graphs for any t). Similarly, the quasipolynomial-time algorithm for $(\mathrm{tw} \leq d, \psi)$-MWIS works for $C_{>t}$-free graphs, for any t. Note that $C_{>t}$-free graphs form a proper superclass of P_t-free graphs. We believe that all polynomial-time results for P_t-free graphs, discussed in this paper, can be actually lifted to $C_{>t}$-free graphs.

References

1 Tara Abrishami, Maria Chudnovsky, Marcin Pilipczuk, Paweł Rzążewski, and Paul D. Seymour. Induced subgraphs of bounded treewidth and the container method. *SIAM J. Comput.*, 53(3):624–647, 2024. doi:10.1137/20M1383732.

2 Benjamin Bergougnoux, Édouard Bonnet, Nick Brettell, and O-joung Kwon. Close relatives of feedback vertex set without single-exponential algorithms parameterized by treewidth. 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 3:1–3:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.IPEC.2020.3.

3 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, Florence, Italy, January 22-25, 2023*, pages 3282–3304. SIAM, 2023. doi:10.1137/1.9781611977554.CH125.

4 Marthe Bonamy, Nicolas Bousquet, Michał Pilipczuk, Paweł Rzążewski, Stéphan Thomassé, and Bartosz Walczak. Degeneracy of P_t-free and $C_{>t}$-free graphs with no large complete bipartite subgraphs. *J. Comb. Theory B*, 152:353–378, 2022. doi:10.1016/J.JCTB.2021.10.005.

5 Flavia Bonomo, Maria Chudnovsky, Peter Maceli, Oliver Schaudt, Maya Stein, and Mingxian Zhong. Three-coloring and list three-coloring of graphs without induced paths on seven vertices. *Comb.*, 38(4):779–801, 2018. doi:10.1007/S00493-017-3553-8.

6 Flavia Bonomo-Braberman, Maria Chudnovsky, Jan Goedgebeur, Peter Maceli, Oliver Schaudt, Maya Stein, and Mingxian Zhong. Better 3-coloring algorithms: Excluding a triangle and a seven vertex path. *Theor. Comput. Sci.*, 850:98–115, 2021. doi:10.1016/J.TCS.2020.10.032.

7 Vincent Bouchitté and Ioan Todinca. Listing all potential maximal cliques of a graph. *Theor. Comput. Sci.*, 276(1-2):17–32, 2002. doi:10.1016/S0304-3975(01)00007-X.

8 Vincent Bouchitté and Ioan Todinca. Treewidth and minimum fill-in: Grouping the minimal separators. *SIAM J. Comput.*, 31(1):212–232, January 2002. doi:10.1137/S0097539799359683.

9 Andreas Brandstädt and Raffaele Mosca. Maximum Weight Independent Sets for (P_7, triangle)-free graphs in polynomial time. *Discret. Appl. Math.*, 236:57–65, 2018. doi:10.1016/J.DAM.2017.10.003.

10 Nick Brettell, Jake Horsfield, Andrea Munaro, Giacomo Paesani, and Daniël Paulusma. Bounding the mim-width of hereditary graph classes. *J. Graph Theory*, 99(1):117–151, 2022. doi:10.1002/JGT.22730.

11 Nick Brettell, Jake Horsfield, Andrea Munaro, and Daniël Paulusma. List k-colouring P_t-free graphs: A mim-width perspective. *Inf. Process. Lett.*, 173:106168, 2022. doi:10.1016/J.IPL.2021.106168.

12 Maria Chudnovsky, Jason King, Michał Pilipczuk, Paweł Rzążewski, and Sophie Spirkl. Finding large H-colorable subgraphs in hereditary graph classes. *SIAM J. Discret. Math.*, 35(4):2357–2386, 2021. doi:10.1137/20M1367660.

13 Maria Chudnovsky, Rose McCarty, Marcin Pilipczuk, Michał Pilipczuk, and Paweł Rzążewski. Sparse induced subgraphs in P_6-free graphs. In David P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 5291–5299. SIAM, 2024. doi:10.1137/1.9781611977912.190.

14 Derek G. Corneil, Yehoshua Perl, and Lorna K. Stewart. A linear recognition algorithm for cographs. *SIAM J. Comput.*, 14(4):926–934, 1985. doi:10.1137/0214065.

15 Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique width. In Juraj Hromkovic and Ondrej Sýkora, editors, *Graph-Theoretic Concepts in Computer Science, 24th International Workshop, WG '98, Smolenice Castle, Slovak Republic, June 18-20, 1998, Proceedings*, volume 1517 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 1998. doi:10.1007/10692760_1.

16 Clément Dallard, Martin Milanič, and Kenny Storgel. Treewidth versus clique number. II. Tree-independence number. *J. Comb. Theory B*, 164:404–442, 2024. doi:10.1016/J.JCTB.2023.10.006.

17 Guo Li Ding, Paul Seymour, and Peter Winkler. Bounding the vertex cover number of a hypergraph. *Combinatorica*, 14(1):23–34, March 1994. doi:10.1007/BF01305948.

18 Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965. doi:10.4153/CJM-1965-045-4.

19 Fedor V. Fomin, Ioan Todinca, and Yngve Villanger. Large induced subgraphs via triangulations and CMSO. *SIAM J. Comput.*, 44(1):54–87, 2015. doi:10.1137/140964801.

20 Peter Gartland and Daniel Lokshtanov. Independent set on P_k-free graphs in quasi-polynomial time. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 613–624. IEEE, 2020. doi:10.1109/FOCS46700.2020.00063.

21 Peter Gartland, Daniel Lokshtanov, Tomás Masařík, Marcin Pilipczuk, Michał Pilipczuk, and Paweł Rzążewski. Maximum weight independent set in graphs with no long claws in quasi-polynomial time. In Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, editors, *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024*, pages 683–691. ACM, 2024. doi:10.1145/3618260.3649791.

22 Peter Gartland, Daniel Lokshtanov, Marcin Pilipczuk, Michał Pilipczuk, and Paweł Rzążewski. Finding large induced sparse subgraphs in $C_{>t}$-free graphs in quasipolynomial time. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 330–341. ACM, 2021. doi:10.1145/3406325.3451034.

23 Andrzej Grzesik, Tereza Klimošová, Marcin Pilipczuk, and Michał Pilipczuk. Polynomial-time algorithm for maximum weight independent set on P_6-free graphs. *ACM Trans. Algorithms*, 18(1):4:1–4:57, 2022. doi:10.1145/3414473.

24 Martin Grötschel, Laszlo Lovász, and Alexander Schrijver. Polynomial algorithms for perfect graphs. In C. Berge and V. Chvátal, editors, *Topics on Perfect Graphs*, volume 88 of *North-Holland Mathematics Studies*, pages 325–356. North-Holland, 1984. doi:10.1016/S0304-0208(08)72943-8.

25 Frédéric Havet, Ross J. Kang, and Jean-Sébastien Sereni. Improper coloring of unit disk graphs. *Networks*, 54(3):150–164, 2009. doi:10.1002/net.20318.

26 Bart M. P. Jansen, Daniel Lokshtanov, and Saket Saurabh. A near-optimal planarization algorithm. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1802–1811. SIAM, 2014. doi:10.1137/1.9781611973402.130.

27 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger, editors, *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, and sponsored by the Office of Naval Research, Mathematics Program, IBM World Trade Corporation, and the IBM Research Mathematical Sciences Department*, pages 85–103, Boston, MA, 1972. Springer US. doi:10.1007/978-1-4684-2001-2_9.

28 Ton Kloks, Ching-Hao Liu, and Sheung-Hung Poon. Feedback vertex set on chordal bipartite graphs. *CoRR*, abs/1104.3915, 2012. doi:10.48550/arXiv.1104.3915.

29 Paloma T. Lima, Martin Milanič, Peter Mursic, Karolina Okrasa, Pawel Rzążewski, and Kenny Štorgel. Tree decompositions meet induced matchings: beyond Max Weight Independent Set. *CoRR*, abs/2402.15834, 2024. doi:10.48550/arXiv.2402.15834.

30 Daniel Lokshtanov, Martin Vatshelle, and Yngve Villanger. Independent set in P_5-free graphs in polynomial time. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 570–581. SIAM, 2014. doi:10.1137/1.9781611973402.43.

31 Vadim V. Lozin and Dieter Rautenbach. Some results on graphs without long induced paths. *Inf. Process. Lett.*, 88(4):167–171, 2003. doi:10.1016/J.IPL.2003.07.004.

32 Vadim V. Lozin and Viktor Zamaraev. The structure and the number of P_7-free bipartite graphs. *Eur. J. Comb.*, 65:143–153, 2017. doi:10.1016/J.EJC.2017.05.008.

33 Pranabendu Misra, Venkatesh Raman, M. S. Ramanujan, and Saket Saurabh. Parameterized algorithms for even cycle transversal. In Martin Charles Golumbic, Michal Stern, Avivit Levy, and Gila Morgenstern, editors, *Graph-Theoretic Concepts in Computer Science - 38th International Workshop, WG 2012, Jerusalem, Israel, June 26-28, 2012, Revised Selcted Papers*, volume 7551 of *Lecture Notes in Computer Science*, pages 172–183. Springer, 2012. doi:10.1007/978-3-642-34611-8_19.

34 Giacomo Paesani, Daniël Paulusma, and Paweł Rzążewski. Feedback Vertex Set and Even Cycle Transversal for H-free graphs: Finding large block graphs. *SIAM J. Discret. Math.*, 36(4):2453–2472, 2022. doi:10.1137/22m1468864.

35 Marcin Pilipczuk. A tight lower bound for Vertex Planarization on graphs of bounded treewidth. *Discret. Appl. Math.*, 231:211–216, 2017. doi:10.1016/j.dam.2016.05.019.

36 Marcin Pilipczuk, Michał Pilipczuk, and Paweł Rzążewski. Quasi-polynomial-time algorithm for independent set in P_t-free graphs via shrinking the space of induced paths. In Hung Viet Le and Valerie King, editors, *4th Symposium on Simplicity in Algorithms, SOSA 2021, Virtual Conference, January 11-12, 2021*, pages 204–209. SIAM, 2021. doi:10.1137/1.9781611976496.23.

37 Marcin Pilipczuk and Paweł Rzążewski. A polynomial bound on the number of minimal separators and potential maximal cliques inP_6-free graphs of bounded clique number. *CoRR*, abs/2310.11573, 2023. doi:10.48550/arXiv.2310.11573.

Covering Weighted Points Using Unit Squares

Chaeyoon Chung
Department of Computer Science and Engineering, Pohang University of Science and Technology, South Korea

Jaegun Lee
Department of Computer Science and Engineering, Pohang University of Science and Technology, South Korea

Hee-Kap Ahn
Department of Computer Science and Engineering, Graduate School of Artificial Intelligence, Pohang University of Science and Technology, South Korea

Abstract

Given a set of n points in d-dimensional space, each assigned a positive weight, we study the problem of finding k axis-parallel unit hypercubes that maximize the total weight of the points contained in their union. In this paper, we present both exact and $(1-\varepsilon)$-approximation algorithms for the case of $k = 2$. We present an exact algorithm that runs in $O(n^2)$ time in the plane, improving the previous $O(n^2 \log^2 n)$-time result. This algorithm generalizes to higher dimensions and larger k in $O(n^{dk/2})$ time for fixed d and k. We also present a $(1-\varepsilon)$-approximation algorithm that runs in $O(n \log \min\{n, 1/\varepsilon\} + 1/\varepsilon^3)$ time for $k = 2$ in the plane, improving the best known result. Our approximation algorithm also extends to higher dimensions.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Maximum coverage, Unit squares, Approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.21

Funding This work was partly supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government(MSIT) (RS-2023-00219980), and the Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.RS-2019-II191906, Artificial Intelligence Graduate School Program (POSTECH)) and (No. 2017-0-00905, Software Star Lab (Optimal Data Structure and Algorithmic Applications in Dynamic Geometric Environment)).

1 Introduction

The maximum coverage problem is a classic and widely studied problem in theoretical computer science and computational geometry. Given a set of weighted elements, a collection of subsets, and an integer k, the goal is to choose up to k subsets that maximize the total weight of the elements contained in the union of the subsets. This problem naturally models many real-world scenarios such as facility location, sensor placement, and resource allocation. It is well-known that the maximum coverage problem is NP-hard and cannot be approximated within a factor better than $(1 - 1/e)$ unless P = NP [5, 6, 13].

In this paper, we study a geometric version of the problem, specifically the variant where the covering objects are axis-parallel unit hypercubes.

▶ **Definition 1** (k-Cover(P)). *Given a set P of n points in d-dimensional space, where each point is assigned a positive weight, find k axis-parallel unit hypercubes such that the total weight of points covered by their union is maximized.*

The problem k-Cover(P) is NP-hard when k is part of the input, even for $d = 2$ [10]. There are results for fixed and small k. For 1-Cover(P), Imai and Asano [7], and Nandy and Bhattacharya [12] gave exact algorithms which run in $O(n \log n)$ time. This is proven to

© Chaeyoon Chung, Jaegun Lee, and Hee-Kap Ahn;
licensed under Creative Commons License CC-BY 4.0

be optimal in the algebraic decision tree model [1]. For 2-Cover(P), Mahapatra et al. [9] gave an algorithm that runs in $O(n^2 \log^2 n)$ time using $O(n \log n)$ space. They also studied a variant where the solution pair of squares must not share a common input point and gave an $O(n^2)$-time algorithm. Cabello et al. [2] considered a more restricted version of 2-Cover(P) where the solution pair of squares must be disjoint and gave an $O(n \log n)$-time algorithm.

There are also approximation algorithms for this problem. For 1-Cover(P), Tao et al.[14] gave a randomized approximation scheme that computes a $(1-\varepsilon)$-approximate solution with probability at least $1 - 1/n$ and runs in $O(n \log(1/\varepsilon)) + n \log \log n)$ time. This was later improved by Jin et al.[8] with an algorithm with running time $O(n \log(1/\varepsilon))$.

For $k \geq 3$, de Berg et al. [4] gave an algorithm that can be extended to compute a $(1-\varepsilon)$-approximate solution to k-Cover(P) in $O(n(k/\varepsilon)^{O(\sqrt{k})})$ time. Jin et al. [8] gave an $(1-\varepsilon)$-approximation algorithm for k-Cover(P) which runs in $O((n/\varepsilon)\log(1/\varepsilon) + (k/\varepsilon)\log(1/\varepsilon) + k(1/\varepsilon)^{\Delta})$ time, where $\Delta = O(\min(\sqrt{k}, 1/\varepsilon))$.

There are work on the problem in higher dimensions $d \geq 3$. For $k = 1$, the problem reduces to the well-known maximum depth problem that, given a set of weighted hypercubes, finds a point maximizing the total weight of the hypercubes containing the point. Chan [3] gave an $O(n^{d/2})$-time algorithm for this problem when $d \geq 3$. For $k \geq 2$, however, little is known in higher dimensions ($d \geq 3$). Mostafavi and Hamzeh [11] studied the problem of approximating the smallest axis-parallel box that covers a given set of points in $\mathbb{R}^d$ while allowing z outliers, and gave a 2-approximation algorithm with running time $O(nd \log n)$.

1.1 Our results

We present both exact and approximation algorithms for 2-Cover(P). We present an $O(n^2)$-time algorithm for 2-Cover(P) in the plane by reducing the problem to the Depth problem (See Section 1.2). Our approach extends for any fixed $d, k > 2$ with little modification, resulting in an $O(n^{dk/2})$-time algorithm for k-Cover(P) in d-dimensional space. More importantly, we present a $(1-\varepsilon)$-approximation algorithm for 2-Cover(P) in the plane running in $O(n \log \min\{n, 1/\varepsilon\} + 1/\varepsilon^3)$ time. This improves upon the previous best result by Jin et al. [8], which runs in $O((n/\varepsilon)\log(1/\varepsilon) + (1/\varepsilon)^{O(1)})$ time, with big hidden constants. In contrast, our algorithm runs faster, is simple, and easy to implement. Our algorithm extends for any fixed $d \geq 3$ with little modification, running in $O((n/\varepsilon^{d-2})\log(1/\varepsilon) + 1/\varepsilon^{2d-1})$ time.

Hidden Constants in the running time of the algorithm in [8]. For $k = 2$, it takes $O((n/\varepsilon)\log(1/\varepsilon) + (1/\varepsilon)^{O(1)})$ time. It begins by constructing a grid G with cells of size $6/\varepsilon$. For each cell c of G, it first computes a $(1-\varepsilon)$-approximate solution to 1-Cover(P_c), where P_c denotes the set of points in P contained in c. Among all cells in G, it greedily selects two cells whose $(1-\varepsilon)$-approximate solution to 1-Cover(P_c) covers the maximum weight. To approximate 2-Cover within each cell, it constructs an additional finer grid consisting of $O(1/\varepsilon^3)$ horizontal and vertical lines, yielding $O(1/\varepsilon^6)$ grid points. Each grid point is considered as a potential candidate for the upper-left corner of a square. Furthermore, this process is repeated over $O(1/\varepsilon)$ different grids. Thus, the second term is $\Omega(1/\varepsilon^7)$ time.

1.2 Preliminaries

For simplicity, we omit the terms "unit" and "axis-parallel" when they are clear from the context. For a compact set X, we use $\mathsf{int}(X)$ to denote the interior of X.

We denote by P a set of positively weighted points in d-dimensional space, and by $w(P)$ the total weight of points in P. We say that a region R *covers* weight w if the total weight of the points in P contained in R is w. For any region R, we denote by $w(R)$ the total weight of points in P covered by R. Let $\mathsf{opt}(P)$ denote the total weight of points covered by an optimal pair of hypercubes to 2-Cover(P).

▶ **Definition 2** (k-Depth(R))**.** *Given a set R of n positively weighted boxes (hyperrectangles) in d-dimensional space, find k points in $\mathbb{R}^d$ that maximizes the sum of weights of the boxes in R containing at least one of the points. Let Depth(R) denote 1-Depth(R).*

2 Exact algorithm

We show that an instance of 2-Cover in the plane can be reduced to an instance of Depth in 4-dimensional space. This yields an $O(n^2)$-time algorithm for 2-Cover in the plane, improving upon the previously best-known $O(n^2 \log^2 n)$-time algorithm. We then generalize this approach to higher dimensions and larger k, reducing an instance of k-Cover in d-dimensional space to an instance of Depth in dk-dimensional space, which can be solved in $O(n^{dk/2})$ time for $d \geq 3$ [3].

2.1 Two squares in the plane

For a set P of n points in the plane, we give a reduction from 2-Cover(P) to an instance of Depth in 4-dimensional space. For a unit square s in the plane, we represent its location by the coordinates of its top-right corner, denoted by (s_x, s_y) (See Figure 1(a)). By representing a pair of unit squares (s, s') as a point in 4-dimensional space, (s_x, s_y, s'_x, s'_y), there is a one-to-one correspondence between a placement of a pair of squares and a point in 4-dimensional space.

A unit square s covers a point $p = (p_x, p_y)$ in the plane if and only if $p_x \leq s_x \leq p_x + 1$ and $p_y \leq s_y \leq p_y + 1$. Let $H(p)$ denote the set of points in 4-dimensional space that correspond to all placements of two unit squares containing p in their union. Then, $H(p) = H_1(p) \cup H_2(p)$ and $H_i(p) = \{(x_1, y_1, x_2, y_2) \mid p_x \leq x_i \leq p_x + 1, p_y \leq y_i \leq p_y + 1\}$ for $i = 1, 2$. See Figure 1.

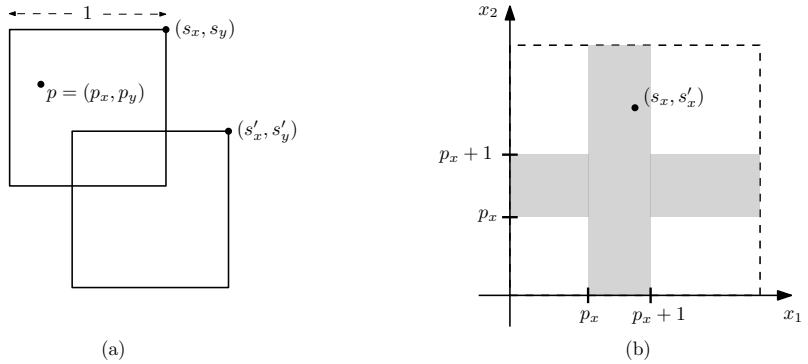


Figure 1 (a) A point $p = (p_x, p_y)$ and a pair of squares (s, s') whose union covers p. (b) A bounding box B is represented by dashed lines. For illustration, we show the projection of $H(p) \cap B$ onto the x_1x_2-plane (gray region). The pair of unit squares (s, s') corresponds to a point (s_x, s_y, s'_x, s'_y) in the 4-dimensional parameter space, and its projection is also shown.

By restricting the parameter space to a sufficiently large bounding box $B = [b_{\min}, b_{\max}]^4$, $H(p)$ can be represented as a union of interior-disjoint boxes in B. (Let $b_{\min}$ be the minimum coordinate value over all points in P and let $b_{\max}$ be the maximum coordinate value over all

points in P plus one.) Clearly, $H(p) \cap B$ can be represented as the union of $O(1)$ mutually interior-disjoint boxes in the 4-dimensional space. Let $B(p)$ denote this set of interior-disjoint boxes such that the weight of each box is set to the weight of p. Let T denote the collection of the boxes in $B(p)$ for all $p \in P$. We now show that an optimal point for $\mathsf{Depth}(T)$ corresponds to an optimal pair of squares for 2-$\mathsf{Cover}(P)$. Here, we slightly abuse the notation so that $\mathsf{opt}(T)$ denotes the total weight of boxes in T containing an optimal point of $\mathsf{Depth}(T)$.

▶ **Lemma 3.** *$opt(T) = opt(P)$.*

Proof. Let (s, s') be an optimal pair of squares for 2-$\mathsf{Cover}(P)$. This pair corresponds to the point $q_{s,s'} = (s_x, s_y, s'_x, s'_y)$ in the 4-dimensional parameter space. Let $P' \subseteq P$ be the set of points covered by the union of s and s', that is, $w(P') = \mathsf{opt}(P)$. By definition, we have $q_{s,s'} \in H(p)$ for every $p \in P'$. Thus, for each $p \in P'$, there is at least one box in $B(p)$ that contains $q_{s,s'}$, implying $\mathsf{opt}(T) \geq \mathsf{opt}(P)$.

Let $q_{r,r'} = (r_x, r_y, r'_x, r'_y)$ be an optimal point for $\mathsf{Depth}(T)$, and let $T' \subseteq T$ be the set of boxes in T that contain $q_{r,r'}$. (By a symbolic perturbation to the boxes, we can avoid double counting while preserving the maximum depth.) Then the total weight of points $p \in P$ such that $q_{r,r'} \in H(p)$ is exactly $\mathsf{opt}(T)$. This means the pair of squares (r, r') together cover the total weight $\mathsf{opt}(T)$ which is at most $\mathsf{opt}(P)$. ◀

As $|T| = O(n)$ and $\mathsf{Depth}(T)$ can be solved in $O(n^{d/2})$ time [3], we have the following.

▶ **Theorem 4.** *Given a set P of n positively weighted points in the plane, we can compute an optimal pair of unit squares for 2-$\mathsf{Cover}(P)$ in $O(n^2)$ time.*

2.2 Extension to larger k and higher dimensions

We can naturally extend this framework to handle k-Cover(P) for general $k \geq 3$. By representing the placement of k squares can also be specified by their top-right corners, it corresponds to a single point in $2k$-dimensional parameter space. There is a one-to-one correspondence between a point in this $2k$-dimensional space and a specific placement of the k squares in the plane. Then $H(p), H_i(p)$, and $B(p)$ can be defined for a point p analogously. The remaining procedure is exactly the same as in the case of $k = 2$.

▶ **Corollary 5.** *Given a set P of n positively weighted points in the plane, we can compute an optimal set of k unit squares for k-$\mathsf{Cover}(P)$ in $O(n^k)$ time.*

Our approach also extends to higher dimensions with little modification, except that the location of a hypercube, represented by one of its corners, now requires d coordinates. As a result, we obtain a set T of boxes in dk-dimensional space.

▶ **Corollary 6.** *Given a set P of n positively weighted points in d-dimensional space, we can compute an optimal set of k unit hypercubes for k-$\mathsf{Cover}(P)$ in $O(n^{dk/2})$ time.*

3 Approximation algorithm

Given a set P of n positively weighted points in the plane, we present a $(1-\varepsilon)$-approximation algorithm for 2-$\mathsf{Cover}(P)$ that runs in $O(n \log \min\{n, 1/\varepsilon\} + 1/\varepsilon^3)$ time for any fixed $\varepsilon > 0$. Specifically, it returns a pair of squares whose union covers a weight at least $(1-\varepsilon) \cdot \mathsf{opt}(P)$. Compared to the $(1-\varepsilon)$-approximation algorithm by Jin et al. [8], our algorithm runs faster asymptotically. Moreover, our algorithm is simple and easy to implement, and it can be extended to higher dimensions with little modifications.

3.1 Types of solutions

For any pair (s_1, s_2) of squares, either (1) $\mathsf{int}(s_1) \cap \mathsf{int}(s_2) = \emptyset$ or (2) $\mathsf{int}(s_1) \cap \mathsf{int}(s_2) \neq \emptyset$. See Figure 2 for an illustration. We solve two cases of the problem: Type (1) case finds two squares (s_1, s_2) with $\mathsf{int}(s_1) \cap \mathsf{int}(s_2) = \emptyset$ whose union covers the maximum weight. Type (2) case finds two squares (s_1, s_2) with $\mathsf{int}(s_1) \cap \mathsf{int}(s_2) \neq \emptyset$ whose union covers the maximum weight. An optimal solution to the type (1) case can be computed in $O(n \log n)$ time [2]. We show how to find a $(1-\varepsilon)$-approximate solution to the type (1) case for 2-Cover(P) in $O(n \log(1/\varepsilon))$ time.

▶ **Lemma 7.** *Given a set P of n positively weighted points in the plane, a $(1-\varepsilon)$-approximate solution to the type (1) case can be computed in $O(n \log(1/\varepsilon))$ time.*

Proof. We note that Jin et al. [8] presented an algorithm that computes a $(1-\varepsilon)$-approximate square for 1-Cover(P) in $O(n \log(1/\varepsilon))$ time. We adopt their grid-shifting technique for our problem. Let $G_3(a, b)$ denote the square grid with mesh size 3, where the vertical and horizontal grid lines are defined as:

$$G_3(a,b) = \left\{(x,y) \in \mathbb{R}^2 \mid x = a + 3k,\ k \in \mathbb{Z}\right\} \cup \left\{(x,y) \in \mathbb{R}^2 \mid y = b + 3k,\ k \in \mathbb{Z}\right\}.$$

Consider the following nine grids: $G_3(a, b)$ for all $a, b \in \{1, 2, 3\}$. It can be easily shown that for any two unit squares s_1 and s_2, there exists at least one of these grids such that both s_1 and s_2 do not intersect any grid lines Therefore, for every grid $G \in \{G_3(a, b) \mid a, b \in \{1, 2, 3\}\}$, we aim to compute a pair of disjoint unit squares that do not intersect any grid lines of G, and we return the pair that covers the maximum total weight.

For a grid $G \in \{G_3(a, b) \mid a, b \in \{1, 2, 3\}\}$, we perform the following preprocessing. We compute a $(1-\varepsilon)$-approximate square for 1-Cover(P_c) for every nonempty cell c of G where P_c denotes the set of points of P in c. We note that Jin et al. [8] give an algorithm, *MaxCovCell*(c), which computes a $(1-\varepsilon)$-approximate to 1-Cover(P_c) in $O(|P_c| \log(1/\varepsilon) + 1/\varepsilon^2)$ time for a cell c of G if the size of c is constant. For each cell of G which contains larger than $(1/\varepsilon)^2$ points, we run *MaxCovCell*. For the remaining nonempty cells, we run the exact algorithm which runs in $O(|P_c| \log |P_c|)$ time [2]. Let $n_1 \geq \ldots n_j \geq (1/\varepsilon^2) \geq n_{j+1} \geq \ldots \geq n_{j+k}$ be the sorted sequence of the number of points in nonempty cells of G. Then the total running time is $\sum_{i=1}^{j} O(n_i \log(1/\varepsilon) + (1/\varepsilon)^2) + \sum_{i=1}^{k} O(n_{i+j} \log(n_{i+j}))$, which is $O(n \log(1/\varepsilon))$.

To make use of the above results, we consider two cases based on whether there exists an optimal disjoint pair of squares for 2-Cover(P) such that the two squares are contained in different cells of G, or the two squares are contained in different cells of G.

Case 1. We first consider the case where there exists an optimal pair of squares of type (1) for 2-Cover(P) such that the two squares are contained in different cells of G. In this case, there always exist at least two cells of G whose respective $(1-\varepsilon)$-approximate solutions together cover at least the weight of $\varepsilon \cdot \mathsf{opt}(P)$. Therefore, among all cells of G, we select the pair of cells whose $(1-\varepsilon)$-approximate solutions together cover the maximum total weight. The union of these two $(1-\varepsilon)$-approximate solutions then forms a $(1-\varepsilon)$-approximate solution for 2-Cover(P).

Before moving onto the second case, we provide a summary of *MaxCovCell(c)*.

MaxCovCell(c). For a given cell c, they form another (non-uniform) grid G' which consists of $O(1/\varepsilon)$ horizontal lines and $O(1/\varepsilon)$ vertical lines such that the following property holds: For a unit square q, it holds that $w(r(q) \cap c) \geq \varepsilon \cdot w(q \cap c)$ where $r(q)$ denotes the largest axis-parallel rectangle contained in q whose corners all lie on grid points of G'. They construct such grid G' in $O(|P_c| \log(1/\varepsilon))$ time. The details of the method used to construct G' can be found in Lemmas 1 and 2 of [8].

For each cell c' of G', they calculate the sum of weights of points at the interior, edges, or corners of c' in $O(|P_c| \log(1/\varepsilon))$ time. Then they enumerate every vertex of G' and consider the unit square q which has its top-left corner on the vertex. While enumerating those vertices and corresponding unit squares q, they calculate $w(r(q) \cap c)$. Then the one that covers the maximum weight can be found in $O(1/\varepsilon^2)$ time with a standard incremental algorithm, and it is a $(1-\varepsilon)$-approximate solution to 1-Cover(P_c).

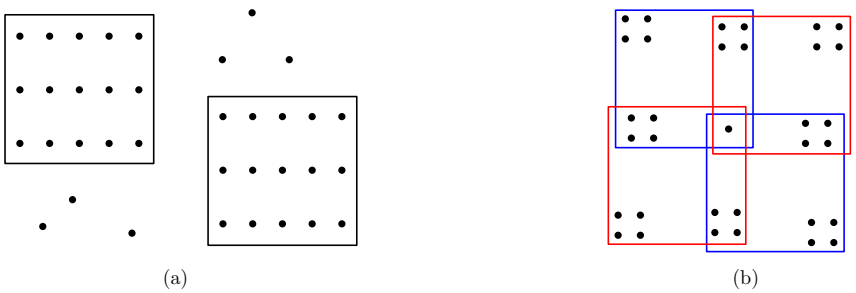
Case 2. Now we consider the case where an optimal pair of squares of type (1) for 2-Cover(P) is contained in the same cell of G. We can observe that once we compute a $(1-\varepsilon)$-approximate pair of squares to 2-Cover(P_c) for every cell c of G, and we can return the pair which covers the maximum total weight as a $(1-\varepsilon)$-approximate solution to 2-Cover(P).

We first note that there exists an $O(n \log n)$-time exact algorithm for 2-Cover(P) in the case where a disjoint optimal pair of squares exists [2]. For any cell c with $|P_c| \leq 1/\varepsilon^2$, we can directly apply this algorithm for 2-Cover(P_c), which takes $O(|P_c| \log |P_c|) = O(|P_c| \log(1/\varepsilon))$ time. Therefore, we now focus on the remaining cells c with $|P_c| > 1/\varepsilon^2$.

To compute a $(1-\varepsilon)$-approximate solution of type (1) for 2-Cover(P_c), we further utilize the information obtained from the procedure *MaxCovCell(c)*. Recall that we calculated $w(r(q) \cap c)$ for the squares q which has its top-left corner on the vertex of G'. Among the rectangle $r(q)$'s, we find a disjoint pair whose union covers the maximum total weight. This can be done using the observation that there always exists a horizontal or vertical line that separates the two disjoint squares. Without loss of generality, we can sweep a horizontal line across the plane and, at each position, track the maximum value of $w(r(q) \cap c)$ achievable by placing q on one side of the sweep line. We repeat this process by sweeping the line in both directions. By combining the results from both sweeps, we can identify two disjoint rectangles among the $r(q)$'s whose union covers the maximum total weight. We note that this procedure can be implemented as part of the $MaxCovCell(c)$. Then a pair of unit squares, each containing one of the rectangles, is a $(1-\varepsilon)$-approximate solution to 2-Cover(P_c).

Among all the cells c of G, the pair of squares for 2-Cover(P_c) that covers the maximum total weight is a $(1-\varepsilon)$-approximate solution to 2-Cover(P).

For each grid $G \in \{G_3(a,b) \mid a,b \in \{1,2,3\}\}$, we compute a $(1-\varepsilon)$-approximate solution for both cases in $O(n \log(1/\varepsilon))$ time and take the one that covers the larger total weight. Then we repeat this process for every grid $G \in \{G_3(a,b) \mid a,b \in \{1,2,3\}\}$, and we return the pair of squares that covers the maximum total weight. ◀



(a) (b)

Figure 2 Solutions for 2-Cover with unit-weighted points. (a) A solution for the type (1) case, which is optimal for 2-Cover. No optimal to the type (2) case is optimal to 2-Cover. (b) Two optimal solutions (red and blue) to the type (2) case, which are optimal for 2-Cover. No optimal to the type (1) case is optimal to 2-Cover.

If an optimal pair of squares for the type (1) case is an optimal pair of squares for 2-Cover(P), we can compute a $(1-\varepsilon)$-approximate solution for 2-Cover(P) in $O(n\log(1/\varepsilon))$ time by Lemma 7. From now on, we assume that every optimal pair of squares for 2-Cover(P) intersect each other in their interior, and we focus on computing a $(1-\varepsilon)$-approximate solution to the type (2) case. Once we obtain a $(1-\varepsilon)$-approximate solution to the type (2) case, we compare it with the approximate solution to the type (1) case and return the one with the larger total weight.

3.2 Search space of the type (2) case

We begin with reducing the search space for 2-Cover(P). We use s^* to denote an optimal square for 1-Cover(P).

▶ **Lemma 8.** *There is an optimal pair of squares for* 2-Cover(P) *such that both squares intersect* int(s^*).

Proof. Suppose that, for an optimal pair (s_1, s_2) for 2-Cover(P), s_2 does not intersect int(s^*). See Figure 3(a). Since s^* is an optimal square for 1-Cover(P), $w(s_1) \leq w(s^*)$. Thus, $w(s_1 \cup s_2) \leq w(s_1) + w(s_2) \leq w(s^*) + w(s_2)$, implying that (s^*, s_2) is an optimal pair of squares for 2-Cover(P). Observe that int(s^*) $\cap$ int(s_2) $= \emptyset$, which contradicts our assumption that every optimal pair of squares for 2-Cover(P) intersect each other in their interiors. ◀

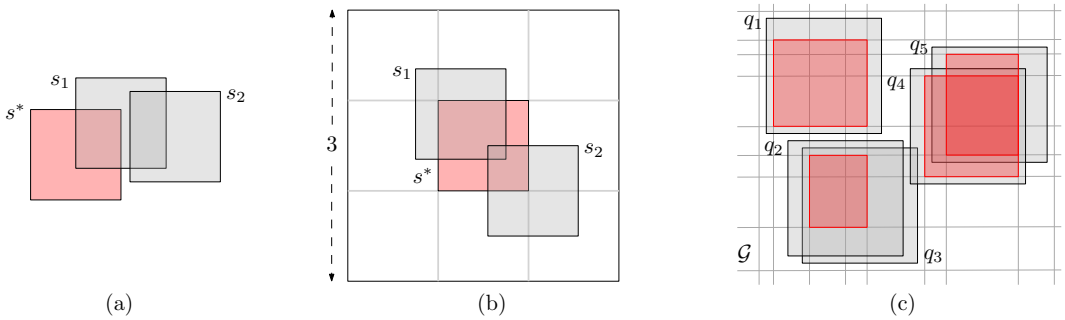


Figure 3 (a) An optimal square s^* for 1-Cover(P) and an optimal pair (s_1, s_2) for 2-Cover(P). (s^*, s_2) is also an optimal pair for 2-Cover(P). (b) 3×3 square centered at s^*. (c) A set $\mathsf{Q} = \{q_1, q_2, \ldots, q_5\}$ of squares and a grid $\mathcal{G}$ formed by L_v and L_h (gray lines). The rectangles in $\mathsf{R} = \{r(q) \mid q \in \mathsf{Q}\}$ are drawn in red. Note that $r(q_2) = r(q_3)$.

Lemma 8 immediately implies the existence of an optimal pair of squares for 2-Cover(P) that are contained in the 3×3 square centered at s^*. See Figure 3(b). We have a similar statement for a $(1-\varepsilon)$-approximate square $\tilde{s}$ for 1-Cover(P).

▶ **Lemma 9.** *There exists a* $(1-\varepsilon)$*-approximate pair of squares* (s_1, s_2) *for* 2-Cover(P) *such that* int(s_1) $\cap$ int($\tilde{s}$) $\neq \emptyset$ *or* int(s_2) $\cap$ int($\tilde{s}$) $\neq \emptyset$.

Proof. Suppose that neither s_1 nor s_2 intersects int($\tilde{s}$). Without loss of generality, assume that $w(s_1) \geq w(s_2)$. Then, $w(s_1) \geq (1-\varepsilon)\mathsf{opt}(P)/2$. Since $\tilde{s}$ is a $(1-\varepsilon)$-approximate square for 1-Cover(P), $2w(\tilde{s}) \geq (1-\varepsilon)\mathsf{opt}(P)$. Thus, $w(s_1 \cup \tilde{s}) = w(s_1) + w(\tilde{s}) \geq (1-\varepsilon)\mathsf{opt}(P)$, implying that $(\tilde{s}, s_1)$ is a $(1-\varepsilon)$-approximate pair of squares for 2-Cover(P). ◀

Recall that we compute a $(1-\varepsilon)$-approximate solution to the type (2) case. By Lemma 9, we have the following.

▶ **Lemma 10.** *There exists a $(1-\varepsilon)$-approximate pair (s_1, s_2) of squares for 2-Cover(P) such that both s_1, s_2 are contained in the 5×5 square centered at $\tilde{s}$.*

By Lemmas 8, 9, and 10, the search space for 2-Cover(P) is a 5×5 square centered at s^* or $\tilde{s}$. We compute either s^* or $\tilde{s}$ depending on the comparison between n and $1/\varepsilon$, identify the corresponding search space, and remove the points of P lying outside this search space. Since s^* can be computed in $O(n \log n)$ time [2] and $\tilde{s}$ can be computed in $O(n \log(1/\varepsilon))$ time by Lemma 7, this step can be completed in $O(n \log \min\{1/\varepsilon, n\})$ time.

From now on, we assume that every point of P lies in the 5×5 square. Clearly, there is a unit square that contains a subset of P whose total weight is at least $w(P)/25$.

▶ **Lemma 11.** *opt$(P) \geq w(P)/25$.*

3.3 Reduction to 2-Depth

Consider a dual mapping between a weighted point and a weighted square. For a point $p = (a, b)$ with weight w_p, the dual $\bar{p}$ of p is a unit square centered at (a, b) and with weight w_p. For a square s with weight w_s centered at (a, b), the dual $\bar{s}$ of s is the point (a, b) with weight w_s. Observe that s contains p if and only if $\bar{p}$ contains $\bar{s}$.

Let Q denote the set of weighted squares $\bar{p}$ for each $p \in P$. Throughout this section, for a square $q \in \mathsf{Q}$, we denote by $w(q)$ the weight q. For an optimal pair (p_1, p_2) of points for 2-Depth(Q), $(\bar{p}_1, \bar{p}_2)$ form an optimal pair of squares for 2-Cover(P). Thus, we transform P into Q, and solve 2-Depth(Q). We then return the pair of squares that is dual to the solution to 2-Depth(Q) as a solution for 2-Cover(P). Since we aims to compute a $(1-\varepsilon)$-approximate solution for 2-Cover(P), it suffices to compute a $(1-\varepsilon)$-approximate solution to 2-Depth(Q). To do this, we construct a grid $\mathcal{G}$ and snap round each rectangle in Q to $\mathcal{G}$.

Grid. We construct a grid $\mathcal{G}$ with $O(1/\varepsilon)$ horizontal and vertical lines with respect to Q using the following lemma, which is obtained by combining Lemmas 1 and 2 of [8].

▶ **Lemma 12** ([8])**.** *Given n positively weighted points with distinct x-coordinates in the plane whose weight sum is W, and a value w_d with $0 < w_d \leq W$, we can find $O(W/w_d)$ vertical lines such that the sum of the weights of points lying in the interior of the slab bounded by any two consecutive lines is at most w_d in time $O(n \log(W/w_d))$.*

Let V_{Q} denote the set of corners of the squares in Q. The weight of $v \in V_{\mathsf{Q}}$ is set to the weight of its corresponding square. Let L_v be the set of vertical lines obtained by applying Lemma 12 with $w_d = \varepsilon \cdot w(P)/50$. Let L_h be the set of horizontal lines defined in exactly the same way. Let $\mathcal{G}$ denote the grid formed by L_v and L_h. By Lemma 12, $\mathcal{G}$ can be constructed in $O(n \log(\min\{n, 1/\varepsilon\}))$ time. In particular, when $n < 1/\varepsilon$, we can simply draw horizontal and vertical lines through every point in V_{Q}.

Snap rounding. For each square $q \in \mathsf{Q}$, let $r(q)$ be the largest axis-parallel rectangle contained in q whose corners all lie on grid points of $\mathcal{G}$. Let $\mathsf{R} = \{r(q) \mid q \in \mathsf{Q}\}$. The weight of $r(q)$ is set to the weight of q. If two or more squares in Q have the same rectangle snap-rounded in $\mathcal{G}$, R contains exactly one snap-rounded rectangle for those squares and the weight of the rectangle is set to the sum of the weights of those squares. See Figure 3(c). For a rectangle $r \in \mathsf{R}$, we denote by $w(r)$ the weight of r. Let opt(R) and opt(Q) denote the total weights obtained by 2-Depth(R) and 2-Depth(Q), respectively.

▶ **Lemma 13.** *opt$(R) \geq (1-\varepsilon) \cdot$ opt(Q).*

Proof. For a point $p \in \mathbb{R}^2$, let $\mathsf{Q}(p) = \{q \in \mathsf{Q} \mid p \in q\}$. Note that there may exist squares $q \in \mathsf{Q}(p)$ with $p \notin r(q)$. Let $\mathsf{Q}^c(p) = \{q \in \mathsf{Q}(p) \mid p \notin r(q)\}$.

Let (p_1, p_2) be an optimal pair of points for 2-$\mathsf{Depth}(\mathsf{Q})$. Assume that p_1 lies in the interior of a vertical slab Γ_v bounded by two consecutive lines in L_v. Also, assume that p_1 lies in the interior of a horizontal slab Γ_h bounded by two consecutive horizontal lines in L_h. Every square in $\mathsf{Q}^c(p_1)$ must have at least two corners lying in $\mathsf{int}(\Gamma_v)$ or at least two corners in $\mathsf{int}(\Gamma_h)$. Since the total weight of the corners in V_{Q} contained in $\mathsf{int}(\Gamma_v)$ or $\mathsf{int}(\Gamma_h)$ is at most $\varepsilon \cdot w(P)/25$, $w(\mathsf{Q}^c(p_1)) \leq \varepsilon \cdot w(P)/50$. Since $\mathsf{opt}(P) = \mathsf{opt}(\mathsf{Q}) \geq w(P)/25$ by Lemma 11, $w(\mathsf{Q}^c(p_1)) \leq \varepsilon \cdot \mathsf{opt}(\mathsf{Q})/2$. The same argument applies to p_2, implying $w(\mathsf{Q}^c(p_2)) \leq \varepsilon \cdot \mathsf{opt}(\mathsf{Q})/2$.

By defining $\mathsf{R}(p)$ analogously to $\mathsf{Q}(p)$, we have $w(\mathsf{R}(p_1) \cup \mathsf{R}(p_2)) \geq w(\mathsf{Q}(p_1) \cup \mathsf{Q}(p_2)) - w(\mathsf{Q}^c(p_1)) - w(\mathsf{Q}^c(p_2))$. Since $w(\mathsf{Q}^c(p_1)) + w(\mathsf{Q}^c(p_2)) \leq \varepsilon \cdot \mathsf{opt}(\mathsf{Q})$ and $w(\mathsf{Q}(p_1) \cup \mathsf{Q}(p_2)) = \mathsf{opt}(\mathsf{Q})$, we have $\mathsf{opt}(\mathsf{R}) \geq (1-\varepsilon) \cdot \mathsf{opt}(\mathsf{Q})$. ◀

For a rectangle $r = [x_1, x_2] \times [y_1, y_2]$, let $x(r) = [x_1, x_2]$ and let $y(r) = [y_1, y_2]$. Let $\mathcal{I}_x = \{x(r) \mid r \in \mathsf{R}\}$ and $\mathcal{I}_y = \{y(r) \mid r \in \mathsf{R}\}$.

▶ **Lemma 14.** *For any two elements $I = [x_1, x_2]$ and $I' = [x'_1, x'_2]$ in $\mathcal{I}_x$, we have $x_2 \leq x'_2$ if $x_1 < x'_1$, and $x_1 \leq x'_1$ if $x_2 < x'_2$. The same property holds for $\mathcal{I}_y$.*

Proof. Let q and q' be the squares in Q with $x(r(q)) = I$ and $x(r(q')) = I'$. Let $x(q) = [a_1, a_2]$ and $x(q') = [a'_1, a'_2]$. Since $x_1 < x'_1$, and q, q' are unit squares, we have $a_1 < a'_1$, and thus $a_2 < a'_2$. This implies that $x_2 \leq x'_2$ by the snap rounding method. The same can be shown for the other cases analogously. ◀

By Lemma 14, we obtain the following corollary. Note that $\mathsf{R} \subseteq \{\alpha \times \beta \mid \alpha \in \mathcal{I}_x, \beta \in \mathcal{I}_y\}$.

▶ **Corollary 15.** *Both $|\mathcal{I}_x|$ and $|\mathcal{I}_y|$ are $O(1/\varepsilon)$. $|R| = O(1/\varepsilon^2)$.*

Recall that $\mathcal{G}$ can be constructed in $O(n \log \min\{n, 1/\varepsilon\})$ time. After sorting the grid lines, we can use binary search to snap round every square in Q in $O(n \log \min\{n, 1/\varepsilon\})$ total time.

▶ **Theorem 16.** *Given a set P of n positively weighted points in the plane, we can compute in $O(n \log \min\{n, 1/\varepsilon\})$ time a set R of positively weighted rectangles in the plane such that $|R| = O(1/\varepsilon^2)$ and the dual of an optimal solution for 2-$\mathsf{Depth}(R)$ is a $(1-\varepsilon)$-approximate solution for 2-$\mathsf{Cover}(P)$.*

3.4 Algorithm

Let R be the set of $O(1/\varepsilon^2)$ positively weighted rectangles obtained by Theorem 16. Recall that each corner of a rectangle in R lies on a grid point of $\mathcal{G}$. As defined in Section 3.3, $\mathcal{G}$ is formed by a set L_v of $O(1/\varepsilon)$ vertical lines and a set L_h of $O(1/\varepsilon)$ horizontal lines. Let $L_v = \{\ell_1, \ell_2, \ldots, \ell_t\}$. For each i from 1 to $t-1$, ℓ_i and ℓ_{i+1}, form a vertical slab which we denote by Γ_i (See Figure 4(a)). Then there is a pair of index (i, j) such that 2-$\mathsf{Depth}(\mathsf{R})$ has an optimal pair of points (p_1, p_2) with $p_1 \in \Gamma_i$ and $p_2 \in \Gamma_j$. For every possible pair of indices (i, j) for $1 \leq i \leq j < t$, we compute an optimal pair of points, (p_1, p_2), for 2-$\mathsf{Depth}(\mathsf{R})$ while restricting that $p_1 \in \Gamma_i$ and $p_2 \in \Gamma_j$. Among all the pairs of points, the one (p_1, p_2) that maximizes the sum of weights of the rectangles in R containing p_1 or p_2 is an optimal solution for 2-$\mathsf{Depth}(\mathsf{R})$.

For two indices $1 \leq i \leq j < t$, we denote by 2-$\mathsf{Depth}(\Gamma_i, \Gamma_j)$, or simply 2-$\mathsf{Depth}(i, j)$, the case of 2-$\mathsf{Depth}(\mathsf{R})$ in which we seek a pair of points (p_1, p_2) maximizing the total weight of rectangles in R containing at least one of them, subject to the constraint that $p_1 \in \Gamma_i$ and $p_2 \in \Gamma_j$.

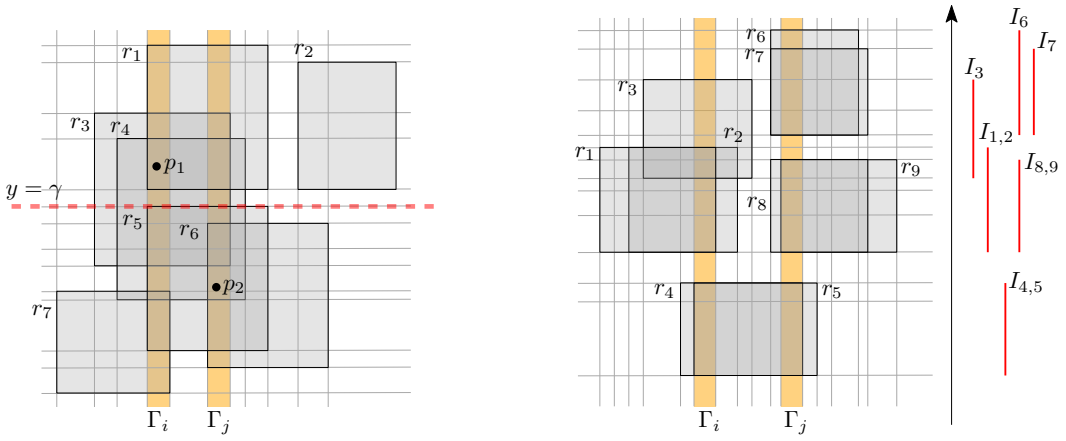


Figure 4 (a) $\mathsf{R}_{i\setminus j} = \{r_7\}$, $\mathsf{R}_{j\setminus i} = \{r_6\}$, $\mathsf{R}_{i,j} = \{r_1, r_3, r_4, r_5\}$. (p_1, p_2) for 2-$\mathsf{Depth}(\mathsf{R})$ with total weight of 5 for unit weight rectangles. $\mathsf{R}_{i,j}(\gamma) = \{r_1, r_3, r_4\}$, $\mathsf{R}^c_{i,j}(\gamma) = \{r_5\}$. $\mathsf{Depth}(\mathsf{R}_{i\setminus j} \cup \mathsf{R}_{i,j}(\gamma)) = 3$, $\mathsf{Depth}(\mathsf{R}_{j\setminus i} \cup \mathsf{R}^c_{i,j}(\gamma)) = 2$. (b) $\mathsf{R}^\ell = [r_1, r_2, r_3, r_4, r_5, r_8, r_7, r_6, r_9]$, $\mathsf{R}^r = [r_1, r_2, r_3, r_4, r_5, r_6, r_8, r_7, r_9]$, $\mathsf{R}^t = [r_4, r_5, r_8, r_9, r_1, r_2, r_3, r_7, r_6]$, $\mathsf{I}_{i\setminus j} = [I_{1,2}, I_3]$, $\mathsf{I}_{j\setminus i} = [I_{8,9}, I_7, I_6]$, $\mathsf{I}_{i,j} = [I_{4,5}]$.

Without loss of generality, we assume that 2-$\mathsf{Depth}(i, j)$ has an optimal pair $(p_1^\star, p_2^\star)$ such that $p_1^\star$ lies to the left of and above $p_2^\star$. The remaining cases can be handled analogously. For ease of description, we assume that $p_1^\star$ and $p_2^\star$ lie in $\mathsf{int}(\Gamma_i)$ and $\mathsf{int}(\Gamma_j)$, respectively. The cases where either p_1 or p_2 lies on the boundary can be handled in a similar manner.

For a rectangle r, let r_y denote the y-coordinate of its top side. For a set H of rectangles and a real value α, let $\mathsf{H}(\alpha) = \{r \in \mathsf{H} \mid \alpha < r_y\}$ and $\mathsf{H}^c(\alpha) = \mathsf{H} \setminus \mathsf{H}(\alpha)$. For a set H of weighted rectangles, let $\mathcal{I}_y(\mathsf{H}) = \{I_y(r) \mid r \in \mathsf{H}\}$. We abuse the notation and use $\mathsf{Depth}(\mathsf{H})$ to denote the total weight of rectangles in H hit by an optimal point for $\mathsf{Depth}(\mathsf{H})$. See Figure 4(a) for the following definition.

▶ **Definition 17.** *Let $R_i = \{r \in R \mid r \cap \mathit{int}(\Gamma_i) \neq \emptyset\}$ for $1 \leq i < t$. For two indices i and j, $1 \leq i < j < t$, let $R_{i\setminus j} = R_i \setminus R_j$, $R_{j\setminus i} = R_j \setminus R_i$, and $R_{i,j} = R_i \cap R_j$. For $i = j$, let $R_{i\setminus i} = \emptyset$ and $R_{i,i} = R_i$.*

For a point $p \in \mathbb{R}^2$, let $x(p)$ and $y(p)$ denote its x and y-coordinates.

▶ **Lemma 18.** *For any fixed i, j with $1 \leq i \leq j < t$, let $\mathsf{opt}(i, j)$ be the total weight of rectangles in R hit by an optimal solution for 2-$\mathit{Depth}(i, j)$. If there is an optimal pair of points (p_1, p_2) for 2-$\mathit{Depth}(i, j)$ with $p_1 \in \Gamma_i, p_2 \in \Gamma_j$ and $y(p_1) > y(p_2)$, there is a real value γ satisfying $\mathit{Depth}(R_{i\setminus j} \cup R_{i,j}(\gamma)) + \mathit{Depth}(R_{j\setminus i} \cup R^c_{i,j}(\gamma)) = \mathsf{opt}(i, j)$.*

Proof. Let $\mathsf{Q}_1 \subset \mathsf{R}$ be the set of rectangles hit by p_1. Let $\mathsf{Q}_2 \subset \mathsf{R}$ be the set of rectangles hit by p_2 but not by p_1. Note that $w(\mathsf{Q}_1) + w(\mathsf{Q}_2) = \mathsf{opt}(i, j)$ as $\mathsf{Q}_1 \cap \mathsf{Q}_2 = \emptyset$.

Let $\gamma = \max\{r_y \mid r \in (\mathsf{Q}_2 \cap \mathsf{R}_{i,j})\}$. See Figure 4(a). Observe that $y(p_2) \leq \gamma < y(p_1)$. We will prove that $w(\mathsf{Q}_1) = \mathsf{Depth}(\mathsf{R}_{i\setminus j} \cup \mathsf{R}_{i,j}(\gamma))$ and $w(\mathsf{Q}_2) = \mathsf{Depth}(\mathsf{R}_{j\setminus i} \cup \mathsf{R}^c_{i,j}(\gamma))$.

Every rectangle $r \in \mathsf{Q}_1$ has $y(p_1) \leq r_y$ by the definition. Every rectangle in Q_1 is from either $\mathsf{R}_{i\setminus j}$ or $\mathsf{R}_{i,j}$. (Note that $\mathsf{R}_{i\setminus j}$ and $\mathsf{R}_{i,j}$ are disjoint.) For a rectangle of the latter case, that is $r \in \mathsf{Q}_1 \cap \mathsf{R}_{i,j}$, observe that $r \in \mathsf{R}_{i,j}(\gamma)$ as we have $\gamma < y(p_1) \leq r_y$. Therefore $\mathsf{Q}_1 \subset (\mathsf{R}_{i\setminus j} \cup \mathsf{R}_{i,j}(\gamma))$.

Every rectangle in Q_2 is from either $\mathsf{R}_{j\setminus i}$ or $\mathsf{R}_{i,j}$. (Note that $\mathsf{R}_{j\setminus i}$ and $\mathsf{R}_{i,j}$ are disjoint.) For a rectangle of the latter case, that is $r \in \mathsf{Q}_2 \cap \mathsf{R}_{i,j}$, observe that $r_y \leq \gamma$ by the definition of γ. Therefore such r is contained in $\mathsf{R}^c_{i,j}(\gamma)$. Therefore $\mathsf{Q}_2 \subset (\mathsf{R}_{j\setminus i} \cup \mathsf{R}^c_{i,j}(\gamma))$.

Now we first show that $w(\mathsf{Q}_1) = \mathsf{Depth}(\mathsf{R}_{i\setminus j} \cup \mathsf{R}_{i,j}(\gamma))$. As $\mathsf{Q}_1 \subset (\mathsf{R}_{i\setminus j} \cup \mathsf{R}_{i,j}(\gamma))$, it is clear that $w(\mathsf{Q}_1) \leq \mathsf{Depth}(\mathsf{R}_{i\setminus j} \cup \mathsf{R}_{i,j}(\gamma))$. Now suppose $w(\mathsf{Q}_1) < \mathsf{Depth}(\mathsf{R}_{i\setminus j} \cup \mathsf{R}_{i,j}(\gamma))$. As every rectangle in $(\mathsf{R}_{i\setminus j} \cup \mathsf{R}_{i,j}(\gamma))$ intersects $\mathsf{int}(\Gamma_i)$, there always exists an optimal point p' for $\mathsf{Depth}(\mathsf{R}_{i\setminus j} \cup \mathsf{R}_{i,j}(\gamma))$ such that $p' \in \Gamma_i$. Let R' be the set of rectangles in $(\mathsf{R}_{i\setminus j} \cup \mathsf{R}_{i,j}(\gamma))$ hit by p', that is $w(\mathsf{R}') = \mathsf{Depth}(\mathsf{R}_{i\setminus j} \cup \mathsf{R}_{i,j}(\gamma))$. Observe that $\mathsf{R}' \cap \mathsf{Q}_2 = \emptyset$ because we have $\mathsf{R}' \subset (\mathsf{R}_{i\setminus j} \cup \mathsf{R}_{i,j}(\gamma))$, $\mathsf{Q}_2 \subset (\mathsf{R}_{j\setminus i} \cup \mathsf{R}^c_{i,j}(\gamma))$, and $(\mathsf{R}_{i\setminus j} \cup \mathsf{R}_{i,j}(\gamma)) \cap (\mathsf{R}_{j\setminus i} \cup \mathsf{R}^c_{i,j}(\gamma)) = \emptyset$. Then the total weight of rectangles in R hit by a new pair (p', p_2) is $w(\mathsf{R}')+w(\mathsf{Q}_2) > w(\mathsf{Q}_1)+w(\mathsf{Q}_2)$, which contradicts that (p_1, p_2) is an optimal pair of points for 2-$\mathsf{Depth}(i, j)$.

In a similar way, we can show that $w(\mathsf{Q}_2) = \mathsf{Depth}(\mathsf{R}_{j\setminus i} \cup \mathsf{R}^c_{i,j}(\gamma))$. ◀

3.4.1 Data Structures and Invariants

Given an input set R of weighted rectangles, we store them in three lists. The first list, R^ℓ, stores the rectangles in increasing order of the x-coordinates of their left sides. The second list, R^r, stores the rectangles in increasing order of the x-coordinates of their right sides. In the case of a tie, the rectangles are further sorted by increasing order of the y-coordinates of their top sides. If the tie still remains unsolved, we sort them by increasing order of the y-coordinates of their bottom sides. See Figure 4(b). The third list, R^t, stores the rectangles in increasing order of the y-coordinates of their top sides. In the case of a tie, the rectangles are further sorted by increasing order of the y-coordinates of their bottom sides. If the tie still remains unsolved, we sort them by increasing order of the x-coordinates of their right sides.

We compute R^ℓ and R^r once at the beginning of the algorithm. Since $|\mathsf{R}| = O(1/\varepsilon^2)$ by Corollary 15, this can be done in $O((1/\varepsilon^2)\log(1/\varepsilon))$ time. For any fixed indices i and j with $1 \leq i \leq j < t$, recall that $\mathcal{I}_y(\mathsf{R}_{i\setminus j})$ denotes the set of weighted intervals induced by $\mathsf{R}_{i\setminus j}$ in the y-direction. By Lemma 15, there are $O(1/\varepsilon)$ intervals in $\mathcal{I}_y(\mathsf{R}_{i\setminus j})$.

Note that two or more rectangles in $\mathsf{R}_{i\setminus j}$ may have the same interval, ignoring their weights. In such a case, we merge them into a single interval. The weight of the merged interval is set to the total weight of those intervals. Let $\mathsf{I}_{i\setminus j}$ denote the resulting set of intervals, which satisfies $|\mathsf{I}_{i\setminus j}| = O(1/\varepsilon)$. In the same way, we define $\mathsf{I}_{j\setminus i}$ and $\mathsf{I}_{i,j}$ with respect to $\mathcal{I}_y(\mathsf{R}_{j\setminus i})$ and $\mathcal{I}_y(\mathsf{R}_{i,j})$, respectively. We always maintain the lists $\mathsf{I}_{i\setminus j}$, $\mathsf{I}_{j\setminus i}$, and $\mathsf{I}_{i,j}$ in increasing order of the right endpoints of the intervals. In the case of a tie, the intervals are further sorted by their left endpoints.

3.4.2 Algorithm for fixed indices i, j

We consider two fixed indices i and j with $1 \leq i \leq j < t$, and present an algorithm that finds an optimal pair of points for 2-$\mathsf{Depth}(i, j)$. For now, we assume that the sorted lists $\mathsf{I}_{i\setminus j}$, $\mathsf{I}_{j\setminus i}$, and $\mathsf{I}_{i,j}$ are given. We will later show how to compute them efficiently. By Lemma 18, there is a real value γ such that an optimal point $p_1 \in \Gamma_i$ for $\mathsf{Depth}(\mathsf{R}_{i\setminus j} \cup \mathsf{R}_{i,j}(\gamma))$, and an optimal point $p_2 \in \Gamma_j$ for $\mathsf{Depth}(\mathsf{R}_{j\setminus i} \cup \mathsf{R}^c_{i,j}(\gamma))$ together form an optimal pair (p_1, p_2) for 2-$\mathsf{Depth}(i, j)$. Thus, our goal is to find such γ and the corresponding optimal pair (p_1, p_2).

Since every rectangle in $\mathsf{R}_{i\setminus j} \cup \mathsf{R}_{i,j}(\gamma)$ intersects $\mathsf{int}(\Gamma_i)$, $\mathsf{Depth}(\mathsf{R}_{i\setminus j} \cup \mathsf{R}_{i,j}(\gamma))$ can be reduced to the one-dimensional problem $\mathsf{Depth}(\mathcal{I}_y(\mathsf{R}_{i\setminus j}) \cup \mathcal{I}_y(\mathsf{R}_{i,j}(\gamma)))$. Recall that for any fixed point p on the real line, the total weight of intervals in $\mathcal{I}_y(\mathsf{R}_{i\setminus j}) \cup \mathcal{I}_y(\mathsf{R}_{i,j}(\gamma))$ hit by p is the same as the total weight of intervals in $\mathsf{I}_{i\setminus j} \cup \mathsf{I}_{i,j}(\gamma)$ hit by p. (Here, we slightly abuse the notation: For a list I of intervals on the real line and a real value α, let $\mathsf{I}(\alpha)$ denote the set of intervals in I whose right endpoints are larger than α, and let $\mathsf{I}^c(\alpha)$ denote the remaining intervals in I. Let $\mathsf{I} \cup \mathsf{I}'$ denotes the set consisting of all intervals from I and I'.) Therefore, for any fixed γ, it suffices to compute $\mathsf{Depth}(\mathsf{I}_{i\setminus j} \cup \mathsf{I}_{i,j}(\gamma))$. Similarly, $\mathsf{Depth}(\mathsf{R}_{j\setminus i} \cup \mathsf{R}^c_{i,j}(\gamma))$ is reduced to $\mathsf{Depth}(\mathsf{I}_{j\setminus i} \cup \mathsf{I}^c_{i,j}(\gamma))$.

Thus, our objective is to find a real value γ that maximizes $\mathsf{Depth}(\mathsf{I}_{i\setminus j} \cup \mathsf{I}_{i,j}(\gamma)) + \mathsf{Depth}(\mathsf{I}_{j\setminus i} \cup \mathsf{I}^c_{i,j}(\gamma))$. Imagine that γ decreases from ∞. Then $\mathsf{Depth}(\mathsf{I}_{i\setminus j} \cup \mathsf{I}_{i,j}(\gamma))$ does not decrease, and we can easily maintain this using a simple incremental update since the intervals in $\mathsf{I}_{i\setminus j} \cup \mathsf{I}_{i,j}$ satisfy the property described in Lemma 14. For $\mathsf{Depth}(\mathsf{I}_{j\setminus i} \cup \mathsf{I}^c_{i,j}(\gamma))$, we increase γ from $-\infty$, and the update can be done in a similar manner. Since all the lists are already sorted, this entire process takes linear time, and we can find the value of γ that maximizes $\mathsf{Depth}(\mathsf{I}_{i\setminus j} \cup \mathsf{I}_{i,j}(\gamma)) + \mathsf{Depth}(\mathsf{I}_{j\setminus i} \cup \mathsf{I}^c_{i,j}(\gamma))$. As there are $O(1/\varepsilon)$ intervals in $\mathsf{I}_{i\setminus j}$, $\mathsf{I}_{j\setminus i}$, and $\mathsf{I}_{i,j}$, we have the following lemma.

▶ **Lemma 19.** *For any fixed i and j with $1 \le i \le j < t$, once we have $\mathsf{I}_{i\setminus j}$, $\mathsf{I}_{j\setminus i}$, and $\mathsf{I}_{i,j}$, we can solve 2-$\mathsf{Depth}(i,j)$ in $O(1/\varepsilon)$ time.*

3.4.3 Final algorithm

For any fixed i with $1 \le i < t$, we show how to compute 2-$\mathsf{Depth}(i,j)$ for every $j = i, i+1, \ldots, t-1$ in total time $O((1/\varepsilon^2)\log(1/\varepsilon))$. Since $t = O(1/\varepsilon)$, this can be done by running the algorithm from Lemma 19 for each j in the range $(t-i+1) = O(1/\varepsilon)$, resulting in a $O((1/\varepsilon^2)\log(1/\varepsilon))$ time in total.

However, it remains to show that the lists $\mathsf{I}_{i\setminus j}$, $\mathsf{I}_{j\setminus i}$, and $\mathsf{I}_{i,j}$ are properly updated as we enumerate j from i to $t-1$. To this end, we introduce new notations S_i and T_i, and provide inductive relations for $\mathsf{R}_{i\setminus j}$, $\mathsf{R}_{j\setminus i}$, and $\mathsf{R}_{i,j}$.

▶ **Definition 20.** *For an index $1 \le i < t$, let $R_i = \{r \in R \mid r \cap \mathit{int}(\Gamma_i) \neq \emptyset\}$. Let $R_0 = R_t = \emptyset$. We then define $S_i = R_i \setminus R_{i-1}$ and $T_i = R_i \setminus R_{i+1}$ for $1 \le i < t$.*

For any fixed i with $1 \le i < t$, the rectangles of S_i appear as a contiguous sublist in R^ℓ, and they appear before those of S_{i+1} in R^ℓ. Symmetrically, the rectangles of T_i appear as a contiguous sublist in R^r, and they appear before those of T_{i+1} in R^r.

▶ **Observation 21.** *For i and j with $1 \le i \le j < t$, $R_{i\setminus j+1} = R_{i\setminus j} \cup (R_{i,j} \cap T_j)$, $R_{j+1\setminus i} = (R_{j\setminus i} \setminus T_j) \cup S_{j+1}$, and $R_{i,j+1} = R_{i,j} \setminus T_j$.*

Using the inductive relation above, we prove the following lemma.

▶ **Lemma 22.** *For any fixed i with $1 \le i < t$, the lists $\mathsf{I}_{i\setminus j}$, $\mathsf{I}_{j\setminus i}$, and $\mathsf{I}_{i,j}$ for all $j = i, \ldots, t-1$ can be computed in time $O(1/\varepsilon^2)$ in total.*

Proof. For the base case where $i = j$, recall that $\mathsf{R}_{i\setminus i} = \emptyset$ and $\mathsf{R}_{i,i} = \{r \in \mathsf{R} \mid r \cap \Gamma_i \neq \emptyset\}$. Thus, we initialize $\mathsf{I}_{i\setminus j}$ and $\mathsf{I}_{j\setminus i}$ as empty lists. For $\mathsf{I}_{i,i}$, we find the rectangles in $\mathsf{R}_{i,i}$ and store them in increasing order of the y-coordinates of their top sides. This can be done by scanning the list R^t. Note that two or more rectangles in $\mathsf{R}_{i\setminus j}$ may have the same interval, ignoring their weights. In such a case, we merge them into a single interval. The weight of the merged interval is set to the total weight of those intervals. This process takes $O(|\mathsf{R}^t|) = O(1/\varepsilon^2)$ time, producing the correct $\mathsf{I}_{i,j}$.

We maintain two pointers, one each for R^ℓ and R^r. As j increases, the pointer in R^ℓ moves to the first element contained in S_{j+1}. and the pointer in R^r moves to the first element contained in T_j.

Assuming we already have $\mathsf{I}_{i\setminus j}$, by Observation 21, we need to find rectangles in $(\mathsf{R}_{i,j} \cap \mathsf{T}_j)$ to obtain $\mathsf{I}_{i\setminus j+1}$. This can be done by scanning the elements in R^r starting from its pointer, which takes $O(|\mathsf{T}_j|)$ time. Note that these rectangles share the same right-side x-coordinate and are sorted by the y-coordinates of their top sides. Hence, we can convert them into intervals, and merge the same intervals (ignoring their weights) as done in the previous

paragraph. This also takes $O(|\mathsf{T}_j|)$ time. By Lemma 15, the resulting set of intervals, say $\mathsf{I}_{\mathsf{new}}$, has size $O(1/\varepsilon)$. We then merge $\mathsf{I}_{\mathsf{new}}$ into $\mathsf{I}_{i\setminus j}$. We merge the same intervals as done in the previous paragraph if necessary. Since both $\mathsf{I}_{i\setminus j}$ and $\mathsf{I}_{\mathsf{new}}$ are sorted by their right endpoints, this merge can be done in $O(|\mathsf{I}_{i\setminus j}| + |\mathsf{I}_{\mathsf{new}}|) = O(1/\varepsilon)$ time.

Similarly, we need to find the rectangles in $(\mathsf{R}_{j\setminus i} \cap \mathsf{T}_j) \cup \mathsf{S}_{j+1}$ to obtain $\mathsf{I}_{j+1\setminus i}$ from $\mathsf{I}_{j\setminus i}$. The rectangles in $(\mathsf{R}_{j\setminus i} \cap \mathsf{T}_j)$ can be found by scanning the elements in R^r, as before. We then convert these rectangles into intervals and merge the same intervals, which can be done in $O(|\mathsf{T}_j|)$ time. After that, we update $\mathsf{I}_{j\setminus i}$ by removing or decreasing the weights of the corresponding intervals. In the same way, we find the rectangles in S_{j+1} by scanning the elements in R^ℓ from its pointer, convert them into intervals, and process them similarly. Therefore, the total time required is $O(|\mathsf{T}_j| + |\mathsf{S}_{j+1}| + 1/\varepsilon)$. The update for $\mathsf{I}_{i,j+1}$ can be handled in the same manner.

It takes $O(|\mathsf{T}_j|+1/\varepsilon)$ time for computing $\mathsf{I}_{i\setminus j+1}$, $O(|\mathsf{T}_j|+|\mathsf{S}_{j+1}|+1/\varepsilon)$ time for computing $\mathsf{I}_{j+1\setminus i}$, and $O(|\mathsf{T}_j| + 1/\varepsilon)$ time from computing $\mathsf{I}_{i,j+1}$. Summing over $j = i$ to $t-1$, the total time is $\sum_{j=i}^{t-1} O\big(|\mathsf{S}_{j+1}| + |\mathsf{T}_j| + 1/\varepsilon\big) = O(|\mathsf{R}|) + O(1/\varepsilon^2) = O(1/\varepsilon^2)$. ◀

By combining Lemmas 19 and 22, we can conclude that for any fixed index i with $1 \le i < t$, we can compute 2-$\mathsf{Depth}(i, j)$ for all $j = i, \ldots, t-1$ in $O(1/\varepsilon^2)$ total time. Repeating this process for all $i = 1, \ldots, t-1$ takes $O(1/\varepsilon^3)$. Since it takes $O(n \log \min\{n, 1/\varepsilon\})$ time to compute the set R by Theorem 16, we obtain the following result.

▶ **Theorem 23.** *Given a set P of n positively weighted points in the plane, we can compute a $(1-\varepsilon)$-approximate pair of squares for 2-$\mathsf{Cover}(P)$ in $O(n \log \min\{n, 1/\varepsilon\} + 1/\varepsilon^3)$ time.*

3.5 Extension to higher dimensions

Our algorithm extends naturally to higher dimensions. For any fixed $d \ge 3$, an optimal pair to the type (1) case can be computed in $O((n/\varepsilon^{d-2}) \log(1/\varepsilon))$ time by extending the algorithm of Lemma 7.

Consider the type (2) case. As in Section 3.2, the search space can be reduced to a constant-size d-dimensional box, and the same duality transform can also be applied. By applying a lemma analogous to Lemma 12, we can construct a grid such that there are $O(1/\varepsilon)$ grid hyperplanes along each axis x_i for $i = 1, \ldots, d$. After performing snap rounding for each dual hypercube into hyperrectangle, the algorithm proceeds in exactly the same manner as in the two-dimensional case.

Let $\Gamma_1, \ldots, \Gamma_t$ denote the slabs formed by two consecutive grid hyperplanes along the x_1-axis. For any fixed $1 \le i \le j \le t$, we solve 2-$\mathsf{Depth}(i, j)$. We can observe that once i and j are fixed, all the hyperrectangles of our interest intersect either Γ_i or Γ_j. We then project the problem onto the plane orthogonal to the x_1-axis and recursively solve the lower-dimensional instance.

Thus, we obtain the following theorem.

▶ **Theorem 24.** *Given a set P of n positively weighted points in d-dimensional space, we can compute a $(1-\varepsilon)$-approximate pair of hypercubes for 2-$\mathsf{Cover}(P)$ in $O((n/\varepsilon^{d-2}) \log(1/\varepsilon) + 1/\varepsilon^{2d-1})$ time for any fixed $d \ge 3$.*

References

1 Michael Ben-Or. Lower bounds for algebraic computation trees. In *Proceedings of 15th Annual ACM Symposium on Theory of Computing*, pages 80–86, New York, NY, USA, 1983. Association for Computing Machinery.

2 Sergio Cabello, J. Miguel Díaz-Báñez, Carlos Seara, J. Antoni Sellès, Jorge Urrutia, and Inmaculada Ventura. Covering point sets with two disjoint disks or squares. *Computational Geometry*, 40(3):195–206, 2008. doi:10.1016/J.COMGEO.2007.10.001.

3 Timothy M. Chan. Klee's measure problem made easy. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 410–419, 2013. doi:10.1109/FOCS.2013.51.

4 Mark de Berg, Sergio Cabello, and Sariel Har-Peled. Covering many or few points with unit disks. *Theory of Computing Systems*, 45(3):446–469, 2009. doi:10.1007/S00224-008-9135-9.

5 Uriel Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998. doi:10.1145/285055.285059.

6 Dorit S. Hochbaum and Anu Pathria. Analysis of the greedy approach in problems of maximum k-coverage. *Naval Research Logistics*, 45(6):615–627, 1998.

7 Hiroshi Imai and Takao Asano. Finding the connected components and a maximum clique of an intersection graph of rectangles in the plane. *Journal of Algorithms*, 4(4):310–323, 1983. doi:10.1016/0196-6774(83)90012-3.

8 Kai Jin, Jian Li, Haitao Wang, Bowei Zhang, and Ningye Zhang. Near-linear time approximation schemes for geometric maximum coverage. *Theoretical Computer Science*, 725:64–78, 2018. doi:10.1016/J.TCS.2017.11.026.

9 Priya Ranjan Sinha Mahapatra, Partha P. Goswami, and Sandip Das. Placing two axis-parallel squares to maximize the number of enclosed points. *International Journal of Computational Geometry & Applications*, 25(04):263–282, 2015. doi:10.1142/S0218195915500156.

10 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.

11 Ali Mostafavi and Ali Hamzeh. High-dimensional axis-aligned bounding box with outliers. In *Proceedings of 34th Canadian Conference on Computational Geometry*, pages 264–269, 2022.

12 Subhas C. Nandy and Bhargab B. Bhattacharya. A unified algorithm for finding maximum and minimum object enclosing rectangles and cuboids. *Computers & Mathematics with Applications*, 29(8):45–61, 1995.

13 George L. Nemhauser, Laurence Wolsey, and M.L. Fisher. An analysis of approximations for maximizing submodular set functions-I. *Mathematical Programming*, 14(1):265–294, 1978. doi:10.1007/BF01588971.

14 Yufei Tao, Xiaocheng Hu, Dong-Wan Choi, and Chin-Wan Chung. Approximate MaxRS in spatial databases. In *Proceedings of the VLDB Endowment*, pages 1546–1557. VLDB Endowment, 2013. doi:10.14778/2536258.2536266.

Minimum Partition of Polygons Under Width and Cut Constraints

Jaehoon Chung
Korea Institute for Advanced Study (KIAS), Seoul, South Korea

Kazuo Iwama
Department of Industrial Engineering and Engineering Management, National Tsing Hua University, Hsinchu, Taiwan

Chung-Shou Liao
Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan

Hee-Kap Ahn
Graduate School of Artificial Intelligence, Department of Computer Science and Engineering, Pohang University of Science and Technology (POSTECH), South Korea

Abstract

We study the problem of partitioning a polygon into the minimum number of subpolygons using cuts in predetermined directions such that each resulting subpolygon satisfies a given width constraint. A polygon satisfies the unit-width constraint for a set of unit vectors if the length of the orthogonal projection of the polygon on a line parallel to a vector in the set is at most one. We analyze structural properties of the minimum partition numbers, focusing on monotonicity under polygon containment. We show that the minimum partition number of a simple polygon is at least that of any subpolygon, provided that the subpolygon satisfies a certain orientation-wise convexity with respect to the polygon. As a consequence, we prove a partition analogue of the Bang's conjecture about coverings of convex regions in the plane: for any partition of a convex body in the plane, the sum of relative widths of all parts is at least one. For any convex polygon, there exists a direction along which an optimal partition is achieved by parallel cuts. Given such a direction, an optimal partition can be computed in linear time.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Polygon partitioning, Width constraints, Plank problem

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.22

Related Version *Full Version*: https://arxiv.org/abs/2509.09981

Funding *Jaehoon Chung*: Supported by a KIAS Individual Grant AP106101 via the Center for Artificial Intelligence and Natural Sciences at Korea Institute for Advanced Study.
Kazuo Iwama: Supported in part by MOST, Taiwan, under Grants NSTC 110-2223-E-007-001 and NSTC 111-2223-E-007-010.
Chung-Shou Liao: Supported by MOST Taiwan Grants NSTC 111-2221-E-002-207-MY3, 114-2221-E-002-221-MY3, and 114-2221-E-002-220-MY3.
Hee-Kap Ahn: Supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government(MSIT) (RS-2023-00219980) and the Institute of Information & communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No. RS-2019-II191906, Artificial Intelligence Graduate School Program(POSTECH)).

1 Introduction

Most works in partitioning polygons have primarily focused on maximizing geometric measures, such as fatness or the minimum side length of resulting pieces [11, 13, 20]. In this paper, we study an opposite objective: partitioning a polygon into subpolygons whose

© Jaehoon Chung, Kazuo Iwama, Chung-Shou Liao, and Hee-Kap Ahn;
licensed under Creative Commons License CC-BY 4.0

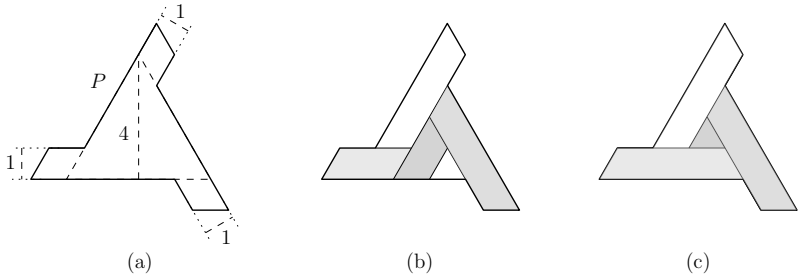


Figure 1 (a) The polygon P has a windmill shape with three arms extending from an equilateral triangle of height 4. (b–c) The minimum partitions of P under constraints $U = \{\mathbf{v}_{0^\circ}, \mathbf{v}_{60^\circ}, \mathbf{v}_{120^\circ}\}$ and $W = \{\mathbf{v}_{30^\circ}, \mathbf{v}_{90^\circ}, \mathbf{v}_{150^\circ}\}$, where $\mathbf{v}_\theta = (\cos\theta, \sin\theta) \in \mathbb{S}^+$. (b) A guillotine partition of five trapezoids. (c) A non-guillotine partition of four trapezoids.

widths are bounded above in certain directions. Such a width constraint commonly arises in manufacturing and recycling industries, where materials must be cut or processed within certain width limits. For example, wood chipping and metal shredding require pieces to fit within the machine's inlet. In some materials, cut directions are critical for preserving structural strength; for example, fabric is typically cut along the fiber direction [17].

Our problem is rooted in classical questions in convex geometry, notably Tarski's plank problem and its affine-invariant extension by Bang [7]. The original conjecture asserts that any covering of a convex body in $\mathbb{R}^d$ by strips must have a total width at least the minimal width of the body, which was proven by Bang. Bang further proposed the *affine plank problem*, in which strip widths are measured relative to the body's width in the same direction. The affine version remains open in general, with only partial results [16, 4, 19, 6]. It is known to be equivalent to the Davenport conjecture [5], which concerns partitions. Several partition analogues have been studied [9, 3, 10].

Our work can also be viewed as a partition analogue of these problems in the plane, where width constraints replace strips, and simple polygons substitute convex bodies.

1.1 Problem definition and results

We consider the problem of partitioning polygons into the minimum number of pieces satisfying both a unit-width constraint and a cut constraint. Let P be a simple polygon. Let Q be a piece in a partition of P satisfying unit-width constraint $W \subseteq \mathbb{S}^+$, where $\mathbb{S}^+$ is the set of unit vectors $\{(\cos\theta, \sin\theta) \mid 0 \le \theta < \pi\}$. Let $\omega_{\mathbf{v}}(Q)$ denote the *width* of Q in $\mathbf{v} \in \mathbb{S}^+$, that is, the length of the orthogonal projection of Q on a line parallel to $\mathbf{v}$. We say Q satisfies the *unit-width constraint* W if $\omega_{\mathbf{v}}(Q) \le 1$ for some vector $\mathbf{v} \in W$.

The process of partitioning P must satisfy a cut constraint $U \subseteq \mathbb{S}^+$. A *cut* is defined as a line segment within P whose relative interior lies in the interior of P. A unit vector is often used to represent a direction, such as the orientation of a cut, meaning that the cut lies on a line parallel to the vector. We require that every cut must be in a direction in U. If a cut extends from one edge of P to the other edge, it divides P into two distinct pieces; such a cut is called a *guillotine* cut. A *guillotine partition* of P (also called a *binary partition*) is obtained by a finite sequence of guillotine cuts; it starts from P and recursively partitions each piece into two subpieces using a guillotine cut. A non-guillotine partition of P is an arbitrary partition of P using cuts that are not necessarily guillotine. Figure 1 shows a guillotine partition and a non-guillotine partition.

Given a simple polygon P, our objective is to partition P into the minimum number of pieces using cuts constrained by U such that each piece in the partition satisfies unit-width constraint W. We denote this minimum partition problem by $\mathsf{wPartition}(P, W, U)$ and its optimum value by $\mathsf{opt}(P, W, U)$. Throughout this paper, we use W and U exclusively to refer to the unit-width constraint and the cut constraint, respectively.

Related Work. Damian and Pemmaraju [14] and Damian-Iordache [15] gave a polynomial-time algorithm for partitioning a simple polygon into the minimum number of subpolygons without using Steiner points such that each subpolygon has diameter at most α, for $\alpha > 0$. Later, Buchin and Selbach [11] showed that this problem becomes NP-hard for polygons with holes. Worman [22] proved NP-completeness for a variant in which each subpolygon must be contained in an axis-aligned square of side length α. Abrahamsen and Stade [2] showed that allowing Steiner points leads to NP-hardness for the partition problem under axis-aligned unit-square containment, even for simple polygons without holes. This marks the first known NP-hardness result of the minimum partition problems for hole-free polygons. Abrahamsen and Rasmussen [1] studied the problem of partitioning simple polygons into the minimum number of pieces such that each piece satisfies a bounded-size constraint (e.g., unit area, perimeter, diameter, or containment within unit disks or squares).

Our Results. Our main contribution is an analysis of the minimum partition number under constraints $W, U \subseteq \mathbb{S}^+$. First, we provide necessary and sufficient conditions for the existence of feasible partitions in $\mathsf{wPartition}(P, W, U)$, along with a decision algorithm for testing feasibility

Second, we study the monotonicity of the minimum partition number under polygon containment $Q \subseteq P$ (Sections 3, 4, and 5). We show that this monotonicity does not hold in general, and identify a sufficient condition based on a restricted-orientation convexity, called $\mathcal{O}$-*convexity*, where $\mathcal{O} \subseteq \mathbb{S}^+$. Theorem 3 states that $\mathsf{opt}(Q, W, U) \leq \mathsf{opt}(P, W, U)$ holds if Q is U-convex with respect to P for guillotine partitions, or $\overline{W}$-convex with respect to P for non-guillotine partitions, where $\overline{W}$ is the set of all unit vectors perpendicular to those in W.

Finally, we prove a partition analogue of Bang's conjecture (Section 6). The statement of Bang's conjecture is as follows: if a convex body $K \subset \mathbb{R}^d$ is covered by strips $H_1, \ldots, H_m$, then $\sum_{i=1}^{m} \inf_{\mathbf{v} \in \mathbb{S}^+} \frac{\omega_{\mathbf{v}}(H_i)}{\omega_{\mathbf{v}}(K)} \geq 1$. Our theorem replaces strip coverings with arbitrary partitions and extends the direction set to any subset $W \subseteq \mathbb{S}^+$.

▶ **Theorem 1** (Bang-Type Partition Analogue). *Let $K \subset \mathbb{R}^2$ be a convex body, and let $K_1 \cup \cdots \cup K_m = K$ be its arbitrary partition. Then, for any subset $W \subseteq \mathbb{S}^+$,*

$$\sum_{i=1}^{m} \inf_{\mathbf{v} \in W} \frac{\omega_{\mathbf{v}}(K_i)}{\omega_{\mathbf{v}}(K)} \geq 1.$$

To the best of our knowledge, this is the first partition analogue that allows non-convex pieces. We also show that, for $U \subseteq \overline{W}$, an optimal partition of a convex polygon can be computed in linear time using equally spaced parallel cuts (See Corollary 17). The omitted proofs will be found in the full version of the paper at `https://arxiv.org/abs/2509.09981`.

2 Preliminaries

Let P be a simple polygon with n vertices in the plane. We assume that the vertices of P are given in a list sorted in counterclockwise order along its boundary. A partition of a simple polygon is a set of connected pieces with pairwise disjoint interiors whose union equals the polygon. The cardinality of a partition is the number of its pieces.

For a set $X \subseteq \mathbb{R}^2$, we denote by ∂X the boundary of X, by $\mathsf{int}(X)$ the interior of X, and by $\mathsf{cl}(X)$ the closure of X. We treat a polygon as the union of its interior and boundary; $\mathsf{cl}(P) = P$, ∂P is the boundary, and $\mathsf{int}(P)$ is the interior.

For a point $p \in \mathbb{R}^2$, let $x(p)$ and $y(p)$ be its x- and y-coordinates, respectively. For any two points p and q in $\mathbb{R}^2$, we use $\overline{pq}$ to denote the line segment connecting p and q with length $|\overline{pq}|$. We call $\overline{pq}$ a cut in P if it lies entirely in the interior of P, excluding its endpoints. If both endpoints lie on ∂P, we call it a guillotine cut.

The inner product of any two vectors $\mathbf{u}$ and $\mathbf{v}$ is denoted by $\langle \mathbf{u}, \mathbf{v} \rangle$. The Euclidean norm of a vector $\mathbf{v}$ is denoted by $\|\mathbf{v}\|$. A vector with norm 1 is called a *unit vector*. We use $\mathbb{S}^+$ to denote the set of unit vectors $\{(\cos\theta, \sin\theta) \mid 0 \leq \theta < \pi\}$. For a subset $V \subseteq \mathbb{S}^+$, we define $\overline{V} = \{\mathbf{v} \in \mathbb{S}^+ \mid \langle \mathbf{u}, \mathbf{v} \rangle = 0 \text{ for some } \mathbf{u} \in V\}$.

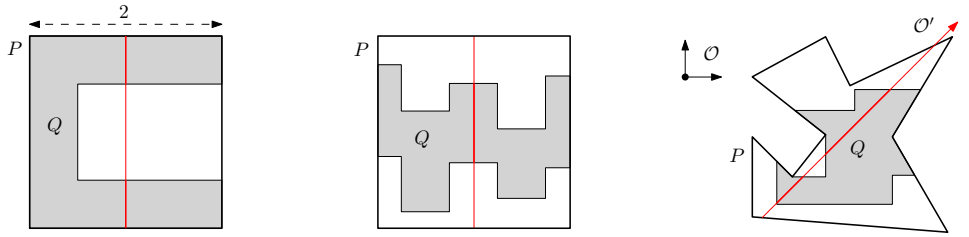
For a compact set $X \subseteq \mathbb{R}^2$ and a vector $\mathbf{v} \in \mathbb{S}^+$, let $\omega_{\mathbf{v}}(X)$ denote the length of the orthogonal projection of X on a line parallel to $\mathbf{v}$. We say X satisfies unit-width constraint $W \subseteq \mathbb{S}^+$ if and only if $\omega_{\mathbf{v}}(X) \leq 1$ for some $\mathbf{v} \in W$.

A *strip* is the region in the plane bounded by two parallel lines. The distance between the bounding lines is the *width* of the strip, and the direction in $\mathbb{S}^+$ orthogonal to the bounding lines is called the *normal* vector of this strip. If a strip has width 1, we call it a *unit* strip.

We use the notation $[m] := \{1, 2, \ldots, m\}$ for a positive integer m. For a finite set A, we use $|A|$ to denote the cardinality of A which is the number of its elements.

3 Monotonicity of minimum partition numbers

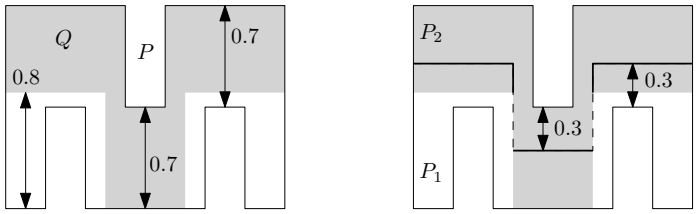
In this section, we assume that $\mathsf{wPartition}(P, W, U)$ has a partition satisfying the constraints. The necessary and sufficient condition for feasibility is presented in the full version. We show $\mathsf{opt}(Q, W, U) \leq \mathsf{opt}(P, W, U)$ for any subpolygon Q of P that satisfies a certain condition. For both guillotine and non-guillotine partitions, we identify sufficient conditions on Q that ensure this monotonicity. When the constraints W and U are clear from context, we abbreviate $\mathsf{opt}(P, W, U)$ as $\mathsf{opt}(P)$.



(a) (b) (c)

Figure 2 (a) A polygon P and its subpolygon Q (gray) with $\mathsf{opt}(P, W, U) = 2$ and $\mathsf{opt}(Q, W, U) > 2$ for $W = \{(1, 0)\}$ and $U = \{(0, 1)\}$. (b) Q (gray) is U-convex (or $\overline{W}$-convex) with respect to P, resulting in $\mathsf{opt}(Q, W, U) \leq \mathsf{opt}(P, W, U)$. (c) Q (gray) of P is $\mathcal{O}$-convex, but not $\mathcal{O}'$-convex with respect to P, where $\mathcal{O} = \{(1, 0), (0, 1)\}$ and $\mathcal{O}' = \{(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2})\}$.

Let P be a square of side length 2, and let Q be a subpolygon of P as shown in Figure 2(a). Consider the instance $\mathsf{wPartition}(P, W, U)$ with $W = \{(1, 0)\}$ and $U = \{(0, 1)\}$. Since U is a singleton, every cut must be a guillotine cut. Observe that neither P nor Q satisfies unit-width constraint W. A vertical cut halving P yields a feasible partition of two pieces. No vertical cut, however, in Q partitions Q into two pieces, each with horizontal width at most 1. Thus, $\mathsf{opt}(Q) > 2 = \mathsf{opt}(P)$, implying that the inclusion $Q \subseteq P$ alone is not sufficient to ensure $\mathsf{opt}(Q) \leq \mathsf{opt}(P)$.



(a) (b)

Figure 3 A non-guillotine partition of P, where $W = \{(0,1)\}$ and $U = \{(0,1),(1,0)\}$. (a) The polygon P and a subpolygon Q, both having vertical widths greater than 1. (b) An optimal non-guillotine partition of P into two pieces, but four when restricted to Q.

Such failures are common in minimum partitioning problems, as the optimal partition number depends on the geometric complexity of Q. Two typical approaches are restricting the geometry of Q relative to P, and constraining the partition class to specific families, such as the guillotine (binary) class [9]. These alter the structural properties of feasible solutions.

3.1 Restricted-orientation convexity

Rawlins [21] introduced the *restricted-orientation convexity* as a generalization of standard convexity in Euclidean space. For $\mathcal{O} \subseteq \mathbb{S}^+$, a set X is *$\mathcal{O}$-convex* if, for every line ℓ parallel to a vector in $\mathcal{O}$, $X \cap \ell$ is connected (we regard an empty set as being connected). When $\mathcal{O} = \mathbb{S}^+$, $\mathcal{O}$-convexity coincides with standard convexity in $\mathbb{R}^2$. We extend this concept to subpolygons of simple polygons with respect to guillotine cuts.

▶ **Definition 2.** *Let Q be a subpolygon of a polygon P, and $\mathcal{O}$ be a set of unit vectors. Then Q is $\mathcal{O}$-convex with respect to P if its intersection with every guillotine cut in P parallel to a vector in $\mathcal{O}$ is connected.*

Figure 2(c) shows a subpolygon Q that is $\mathcal{O}$-convex with respect to P for $\mathcal{O} = \{(0,1),(1,0)\}$, but not $\mathcal{O}'$-convex for $\mathcal{O}' = \{(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2})\}$ since a guillotine cut in P parallel to $(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2})$ intersects Q in at least two connected components.

Observe that $\mathcal{O}$-convexity with respect to a polygon P is equivalent to the *geodesic convexity* within P when $\mathcal{O} = \mathbb{S}^+$. The following theorem gives sufficient conditions for monotonicity of the minimum partition numbers.

▶ **Theorem 3** (Monotonicity in Polygon Containment). *Let U and W be sets of unit vectors, and let Q be a subpolygon of a polygon P. Assume that* wPartition(P, W, U) *has a solution.*

- $\mathsf{opt}(Q) \leq \mathsf{opt}(P)$ *for guillotine partitions if Q is U-convex with respect to P.*
- $\mathsf{opt}(Q) \leq \mathsf{opt}(P)$ *for non-guillotine partitions if Q is $\overline{W}$-convex with respect to P.*

Revisit the subpolygons Q in Figure 2(a,b). In (a), Q is neither U-convex nor $\overline{W}$-convex with respect to P, whereas in (b), Q satisfies both conditions, and Theorem 3 implies that $\mathsf{opt}(Q) \leq \mathsf{opt}(P) = 2$, for both guillotine and non-guillotine partitions.

3.2 Monotonicity in guillotine partitions

We prove Theorem 3 for the guillotine case. Let $\Pi = \{P_1, \ldots, P_m\}$ be a guillotine partition of P feasible to wPartition(P, W, U), and let $\Pi[Q] = \{P_i \cap Q\}_{i=1,\ldots,m}$ be its restriction to a subpolygon Q that is U-convex with respect to P. Then, $\bigcup_{i=1}^{m}(P_i \cap Q) = Q$, and thus, $\Pi[Q]$

also forms a partition of Q. Note that each piece in $\Pi[Q]$ satisfies unit-width constraint W, as every piece is a subset of some P_i that satisfies the constraint. To show $\mathsf{opt}(Q) \leq \mathsf{opt}(P)$, it suffices to verify two aspects: (1) $\Pi[Q]$ is a guillotine partition of Q and (2) $|\Pi[Q]| \leq m$.

Since Π is a guillotine partition, the cuts used for Π are partially ordered by their precedence in the partitioning process. Let $\mathcal{C} = \langle c_1, \ldots, c_{m-1} \rangle$ denote a sequence of guillotine cuts that produces Π. Then each $c_i \cap Q$ is connected as c_i is a cut in P with a direction in U and Q is U-convex with respect to P. Let $\mathcal{D} = \langle d_1, \ldots, d_{m'} \rangle$ be the subsequence of $\langle c_1 \cap Q, \ldots, c_{m-1} \cap Q \rangle$ consisting of those satisfying $c_i \cap \mathsf{int}(Q) \neq \emptyset$ for $i = 1, \ldots, m-1$. Note that each d_j for $j \in [m']$ is a cut in Q and $m' \leq m-1$. Also, observe that $\Pi[Q]$ is induced by $\mathcal{D}$. It suffices to show that $\mathcal{D}$ is indeed a sequence of guillotine cuts in Q: each $d_j \in \mathcal{D}$ is a guillotine cut in one subpolygon obtained by applying $\langle d_1, \ldots, d_{j-1} \rangle$ to Q.

Let i be the smallest index in $[m-1]$ such that $c_i \cap \mathsf{int}(Q) \neq \emptyset$. Since no cut in $\langle c_1, \ldots, c_{i-1} \rangle$ intersects $\mathsf{int}(Q)$, there is a subpolygon in the partition of P by $\langle c_1, \ldots, c_{i_1-1} \rangle$ that contains Q. Observe that c_i is a guillotine cut in this subpolygon that is aligned with a direction in U. Since Q is U-convex with respect to P, $d_1 = c_i \cap Q$ is a line segment, and thus, d_1 is a guillotine cut in Q.

We proceed by induction on j with $1 < j \leq m'$, that $\langle d_1, \ldots, d_{j-1} \rangle$ forms a sequence of guillotine cuts in Q. Among the pieces of Q partitioned by the sequence, let Q_j denote the one containing d_j, where d_j is a cut in Q_j. By definition, there exists an index $k \in [m]$ such that $d_j = c_k \cap Q$. Let P_k be the piece in the partition of P by $\langle c_1, \ldots, c_{k-1} \rangle$ that contains c_k.

To see that d_j is a guillotine cut of Q_j, recall that $d_j = c_k \cap Q_j$, where c_k is a guillotine cut of P_k and $Q_j \subseteq P_k$. Let c'_k be the guillotine cut in P obtained by extending c_k until it touches ∂P. Since Q is U-convex, $c'_k \cap Q$ is connected, and hence so is its subsegment $c_k \cap Q = d_j$. Moreover, as c_k spans ∂P_k, its restriction to Q_j necessarily touches ∂Q_j at both endpoints. Thus, d_j is a guillotine cut in Q_j. We conclude that $\mathcal{D} = \langle d_1, \ldots, d_{m'} \rangle$ induces a guillotine partition of Q. Since $m' \leq m$, we have $|\Pi[Q]| \leq m$.

▶ **Lemma 4.** *Let P be a polygon and let $\Pi = \{P_1, \ldots, P_m\}$ be a solution to the problem $\mathsf{wPartition}(P, W, U)$ for guillotine cuts. If a subpolygon Q of P is U-convex with respect to P, the restricted partition $\{Q \cap P_i\}_{i=1,\ldots,m}$ is a solution to $\mathsf{wPartition}(Q, W, U)$ for guillotine cuts with at most m pieces.*

4 Reconfiguration of restricted non-guillotine partitions

Let Q be a subpolygon of P that is $\overline{W}$-convex with respect to P. The monotonicity $\mathsf{opt}(Q) \leq \mathsf{opt}(P)$ trivially holds when $\mathsf{opt}(Q) = 1$. Also, any feasible partition is guillotine when U is a singleton. Assume that $\mathsf{opt}(Q) > 1$ and U contains at least two distinct vectors.

Let $\Pi = \{P_1, \ldots, P_m\}$ be any feasible partition of P to $\mathsf{wPartition}(P, W, U)$. Its restriction to Q, defined as $\Pi[Q] = \{P_i \cap Q\}_{i=1,\ldots,m}$, is a feasible partition of Q to $\mathsf{wPartition}(Q, W, U)$. However, some regions $P_i \cap Q$ may be disconnected, even when Q is $\overline{W}$-convex with respect to P (See Figure 3(a–b)). To address this, we modify the cuts in $\Pi[Q]$ to reconnect disjoint fragments into connected regions while preserving feasibility for $\mathsf{wPartition}(Q, W, U)$.

Consider any element P_i in the partition Π of P such that $\mathsf{int}(P_i) \cap \mathsf{int}(Q) \neq \emptyset$. The intersection $\mathsf{int}(P_i) \cap \mathsf{int}(Q)$ consists of open connected components. We define R_i as

$$R_i \coloneqq \{\, \mathsf{cl}(X) \mid X \text{ is a connected component of } \mathsf{int}(P_i) \cap \mathsf{int}(Q) \text{ with } \mathsf{cl}(X) \cap \partial Q \neq \emptyset \,\}.$$

Let $R_i = \{C_1, C_2, \ldots, C_t\}$. Note that each C_j is a subpolygon of Q with positive area that touches ∂Q. To connect elements in R_i into a single piece, we reconfigure $\Pi[Q]$ by iteratively performing a process, called *reallocation*. Let X be a connected region in Q. We

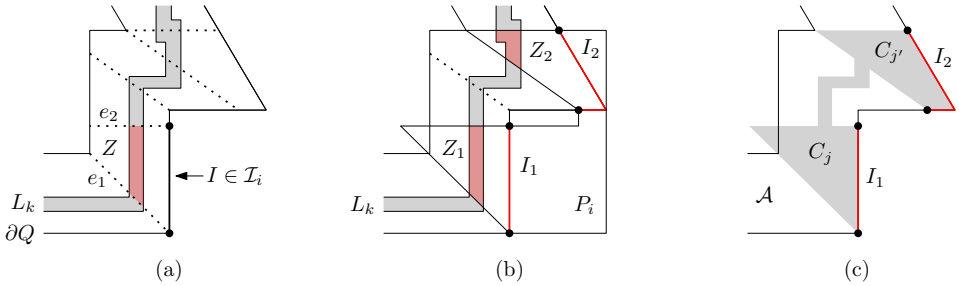


Figure 4 (a) A layer segment Z of R_i is bounded by $e_1, e_2 \in E_i^{\mathrm{bd}}$, and it corresponds to $I \in \mathcal{I}_i$. (b) A link instruction $\lambda = (I_1, I_2, k)$ reallocates the layer segments in $L_k[Z_1, Z_2]$ to P_i. (c) Applying λ merges C_j and $C_{j'}$ into a single piece through layer segments in L_k, where $I_1 \in \mathcal{I}_{ij}$ and $I_2 \in \mathcal{I}_{ij'}$ for some $j, j' \in [t]$.

define the reallocation of X to P_i as the operation that modifies $\Pi[Q]$ by expanding the region assigned to P_i so that it includes X. This can be done by adding new boundaries along $\partial X \setminus P_i$ and removing those along $\partial P_i \setminus X$. If X and P_i intersect in their interiors, or share a boundary segment of positive length, then $X \cup P_i$ appears as a single piece in the resulting partition. We say that the region X is *reallocated* to P_i.

4.1 Construction and layering of corridor of Q

We first construct a narrow corridor along ∂Q. This corridor lies entirely within Q, closely following ∂Q, and provides the space needed to link elements of R_i.

The corridor is an annular region bounded by two simple closed curves: the outer curve ∂Q and an inner curve that lies at a small distance δ inward. The region between the two curves is called the δ-*corridor* of Q, where δ denotes its width. In our construction, we set δ to be a sufficiently small positive value and denote the resulting corridor by $\mathcal{A}$.

Consider the edges of each $C_j \in R_i$ that have one endpoint on ∂Q and intersect $\mathsf{int}(Q)$. Let E_{ij}^{bd} denote the set of such edges of C_j. We define $E_i^{\mathrm{bd}} := \bigcup_{j=1}^{t} E_{ij}^{\mathrm{bd}}$ and $E_Q^{\mathrm{bd}} := \bigcup_{i=1}^{m} E_i^{\mathrm{bd}}$. The corridor $\mathcal{A}$ is subdivided into a sequence of nested subregions, which we refer to as *layers*. Let $h \in \mathbb{Z}_{>0}$ denote the number of layers which will be determined in Section 4.4. For each $k \in [h]$, we construct a simple closed curve Γ_k lying entirely within $\mathsf{int}(\mathcal{A})$, such that the curves $\Gamma_1, \ldots, \Gamma_h$ are pairwise disjoint and arranged sequentially inward from ∂Q. Each Γ_k follows a zigzag pattern using the two directions in U, and intersects every edge in E_Q^{bd} once or twice, depending on whether the edge is a guillotine cut in Q. Details on the construction of the corridor $\mathcal{A}$ and simple curves are provided in the full version.

These h curves subdivide $\mathcal{A}$ into h nested subregions, called *layers*: for each $k \in [h]$, the k-th layer is the annular region bounded by Γ_{k-1} and Γ_k, where we define $\Gamma_0 = \partial Q$.

4.2 Link instructions with circular intervals and layers

Let C_j and $C_{j'}$ be two elements in R_i with $j \neq j'$. For $k \in [h]$, let L_k denote the k-th layer. Each connected component of $L_k \cap C_j$ for $C_j \in R_i$ is a subregion of Q referred to as a *layer segment* of C_j (and of R_i) in L_k. Each layer segment is bounded by four parts: continuous portions of the inner and outer boundaries of L_k, and two edges from E_{ij}^{bd}. Removing any layer segment from L_k alters its topological structure from an annulus to a weakly simple polygon. All layer segments in L_k can be arranged in cyclic order along L_k. Figure 4(a) illustrates a layer segment of R_i in L_k.

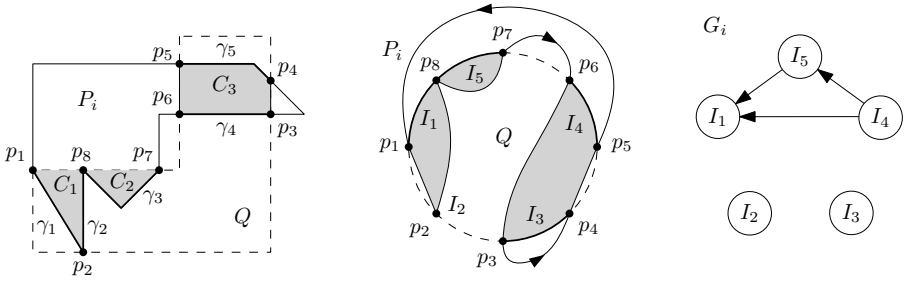


Figure 5 (a) $\mathsf{int}(P_i) \cap \mathsf{int}(Q)$ induces $R_i = \{C_1, C_2, C_3\}$, and each $\partial C_j \cap \mathsf{int}(Q)$ induces simple paths $\{\gamma_1, \ldots, \gamma_5\}$. (b) p_8 serves as both an entry and exit point for the transition $C_1 \to C_2$. The simple paths outside $\mathsf{int}(Q)$, from p_7 to p_6 and from p_5 to p_1, represents the transitions $C_2 \to C_3$ and $C_3 \to C_1$, respectively. (c) The directed graph G_i, constructed based on P_i and Q shown in (a).

We use an interval notation over layer segments that are cyclically ordered. For two layer segments Z_1, Z_2 in L_k, we denote by $L_k[Z_1, Z_2]$ the set of layer segments encountered in counterclockwise traversal from Z_1 to Z_2 in L_k, where $Z_1, Z_2 \in L_k[Z_1, Z_2]$. Similarly, $L_k(Z_1, Z_2)$, $L_k[Z_1, Z_2)$, and $L_k(Z_1, Z_2]$ denote the open and half-open intervals over layer segments between Z_1 and Z_2 in L_k.

Assume that $Z_1 \subseteq C_j$ and $Z_2 \subseteq C_{j'}$ are layer segments of R_i in L_k, for $j, j' \in [t]$ with $j \neq j'$. The reallocation of $L_k[Z_1, Z_2]$ to P_i merges C_j and $C_{j'}$ into a single piece within L_k. We refer to such an ordered pair (Z_1, Z_2) as a *link instruction* of R_i. See Figure 4(b–c).

Decomposition of ∂Q into circular intervals. For $C_j \in R_i$, C_j is a subpolygon of Q that touches ∂Q. The intersection $\partial C_j \cap \partial Q$ consists of continuous paths on ∂Q. Specifically, it can be the loop ∂Q itself if $\partial Q \subseteq \partial C_j$, implying that Q satisfies unit-width constraint W. However, assuming $\mathsf{opt}(Q) > 1$, this case does not occur. Thus, every connected component of $\partial C_j \cap \partial Q$ must be a continuous path on ∂Q that is not a loop. We define $\mathcal{I}_{ij}$ as

$$\mathcal{I}_{ij} \coloneqq \{I \subseteq \partial Q \mid I \text{ is a connected component of } \partial C_j \cap \partial Q\}.$$

We then define the aggregate sets $\mathcal{I}_i \coloneqq \bigcup_{j=1}^{t} \mathcal{I}_{ij}$ and $\mathcal{I}_Q \coloneqq \bigcup_{i=1}^{m} \mathcal{I}_i$.

Let $I \in \mathcal{I}_{ij}$ be a continuous path along ∂Q with endpoints p and q such that I corresponds to the portion of ∂Q from p to q in counterclockwise order. Since ∂Q is a simple closed curve, topologically equivalent to a circle, we regard each path I as a circular interval on ∂Q. As $\mathcal{I}_{ij}$ contains no loops, the case $p = q$ occurs only when ∂C_j touches ∂Q at a single point p, and no other portion of ∂C_j near p intersects ∂Q. Such intervals are called *degenerate* intervals.

For $i \in [m]$, $\mathcal{I}_i$ may contain both degenerate and non-degenerate intervals. Since P_i is simple, all intervals in $\mathcal{I}_i$ are pairwise interior-disjoint. Figure 5(a) shows a piece P_i and the subpolygon Q, from which the circular intervals in $\mathcal{I}_i$ are defined. These intervals are illustrated in Figure 5(b).

Note that a degenerate interval at some point $p \in \partial Q$ may appear in multiple $\mathcal{I}_i$'s. In such cases, each occurrence is treated as a distinct element in $\mathcal{I}_Q$. Since $\Pi = \{P_1, \ldots, P_m\}$ is a partition of P and $Q \subseteq P$, every point on ∂Q belongs to some interval of $\mathcal{I}_Q$, and no two intervals in $\mathcal{I}_Q$ intersect each other in their interiors. Thus, $\mathcal{I}_Q$ forms a decomposition of ∂Q.

Finally, we relate these circular intervals to layer segments so as to define a link instruction formally. For any $j \in [t]$ and any $k \in [h]$, consider the component $C_j \in R_i$ and the layer L_k. Each connected component of $\mathcal{A} \cap C_j$ (or $L_k \cap C_j$) is bounded by four parts: continuous

portions of the inner and outer boundaries of $\mathcal{A}$ (or L_k), and two edges from E_{ij}^{bd}. Each connected component of $L_k \cap C_j$ is entirely contained within a unique connected component of $\mathcal{A} \cap C_j$. Moreover, the portion of ∂Q that bounds each connected component of $\mathcal{A} \cap C_j$ corresponds to a circular interval in $\mathcal{I}_{ij}$. Thus, each layer segment of C_j in L_k is uniquely associated with a circular interval in $\mathcal{I}_{ij}$, in a one-to-one correspondence. The number of layer segments in $\mathcal{A} \cap C_j$ (or $L_k \cap C_j$) is $|\mathcal{I}_{ij}| \cdot h$ (or $|\mathcal{I}_{ij}|$).

We revisit the link instruction that merges two layer segments $Z_1 \subseteq L_k \cap C_j$ and $Z_2 \subseteq L_k \cap C_{j'}$ for $j, j' \in [t]$ along layer L_k. The link instruction reallocates the region in L_k spanned counterclockwise from Z_1 to Z_2. Since Z_1 and Z_2 correspond to circular intervals in $\mathcal{I}_{ij}$ and $\mathcal{I}_{ij'}$, respectively, each link instruction can be represented by a triple (I_1, I_2, k), where $I_1, I_2 \in \mathcal{I}_Q$ and $k \in [h]$ (See Figure 4). Note that $(I_1, I_2, k) \neq (I_2, I_1, k)$, as the counterclockwise span from Z_1 to Z_2 differs from that in the reverse order.

4.3 Graph for encoding link instructions

Recall that $R_i = \{C_1, C_2, \ldots, C_t\}$ is the set of closures of connected pieces of $\mathsf{int}(P_i) \cap \mathsf{int}(Q)$ such that each C_j has a positive area and touches ∂Q. We construct a graph for each R_i, which specifies how the elements of R_i are to be connected into a single piece within Q.

For each $j \in [t]$, $\mathcal{I}_{ij}$ consists of circular intervals, each corresponding to a continuous path in $\partial C_j \cap \partial Q$ along ∂Q. Let $\mathcal{I}_{ij}^+$ denote the subset of $\mathcal{I}_{ij}$ consisting of only those intervals with positive length. We define $\mathcal{I}_i^+ = \bigcup_{j=1}^t \mathcal{I}_{ij}^+$ and $\mathcal{I}_Q^+ = \bigcup_{i=1}^m \mathcal{I}_i^+$. Unlike $\mathcal{I}_Q$, the set $\mathcal{I}_Q^+$ contains only non-degenerate intervals, which are pairwise interior-disjoint. Note that $\{\mathcal{I}_i^+\}_{i=1,\ldots,m}$ forms a partition of ∂Q.

Sequencing subpaths of ∂P_i. For each $C_j \in R_i$, $\partial C_j \cap \mathsf{int}(Q)$ consists of connected components, each forming a continuous path connecting two points on ∂Q. By the definition of $\mathcal{I}_{ij}$, the endpoints of these paths correspond to the endpoints of the circular intervals in $\mathcal{I}_{ij}$. The total number of such paths in $\partial C_j \cap \mathsf{int}(Q)$ is $|\mathcal{I}_{ij}|$. See Figure 5(a).

We traverse ∂P_i counterclockwise, starting at any point on $\partial C_1 \cap \mathsf{int}(Q)$ and completing a full circuit. During the traversal, each path in $\partial C_j \cap \mathsf{int}(Q)$ is visited exactly once for each $j \in [t]$, except for the path containing the starting point. The order in which the continuous paths are visited is denoted by $\gamma_1 \to \gamma_2 \to \cdots \to \gamma_l$, where $\gamma_1 = \gamma_l$ is the path in $\partial C_1 \cap \mathsf{int}(Q)$ containing the starting point, and $l = \sum_{j=1}^t |\mathcal{I}_{ij}| + 1 = |\mathcal{I}_i| + 1$.

Consider two consecutive paths γ_k and γ_{k+1} for any $k \in [l-1]$. Each path is derived from some piece in R_i; there exist $j, j' \in [t]$ such that $\gamma_k \subseteq \partial C_j \cap \mathsf{int}(Q)$ and $\gamma_{k+1} \subseteq \partial C_{j'} \cap \mathsf{int}(Q)$. As we traverse from γ_k to γ_{k+1}, we exit $\mathsf{int}(Q)$ through an endpoint of γ_k and re-enter $\mathsf{int}(Q)$ through an endpoint of γ_{k+1}.

▶ **Lemma 5.** *Let (γ_k, γ_{k+1}) be a pair of consecutive paths for $k \in [l-1]$, where $\gamma_k \subseteq \partial C_j \cap \mathsf{int}(Q)$ and $\gamma_{k+1} \subseteq \partial C_{j'} \cap \mathsf{int}(Q)$ for some $j, j' \in [t]$. The traversal from γ_k to γ_{k+1} exits and re-enters $\mathsf{int}(Q)$ through intervals $I_1 \in \mathcal{I}_{ij}$ and $I_2 \in \mathcal{I}_{ij'}$, respectively. If $j = j'$, then $I_1 = I_2$. Otherwise, I_1 and I_2 are interior-disjoint, non-degenerate intervals.*

By Lemma 5, a pair of distinct intervals in $\mathcal{I}_i^+$ provides an exit-entry pair for each transition between distinct pieces in R_i in the counterclockwise traversal of ∂P_i.

Construction of G_i. Based on the counterclockwise traversal of ∂P_i, we construct a directed graph $G_i = (V_i, E_i)$ as follows. Each vertex $v \in V_i$ corresponds to an interval in $\mathcal{I}_i^+$, so $|V_i| = |\mathcal{I}_i^+|$. Let γ_1 and γ_2 be consecutive paths in the traversal where $\gamma_1 \subseteq \partial C_j \cap \mathsf{int}(Q)$

and $\gamma_2 \subseteq \partial C_{j'} \cap \mathsf{int}(Q)$ with $j \neq j'$. The traversal from γ_1 to γ_2 encounters intervals I_1 and I_2 in $\mathcal{I}_i^+$ that contain the exit and entry points, respectively. By Lemma 5, these intervals I_1 and I_2 are uniquely determined. We add a directed edge e between vertices v_1 and v_2 corresponding to I_1 and I_2, respectively.

Let p and q denote the exit and entry points of the traversal, respectively; p is the endpoint of I_1 and q is the endpoint of I_2. The counterclockwise traversal on ∂P_i between γ_1 and γ_2 follows a simple path along ∂P_i outside Q that starts from p and ends at q. Let $\gamma_{pq} \subseteq \partial P_i \setminus \mathsf{int}(Q)$ denote this path from p to q. Observe that the exit and entry points may coincide, i.e., $p = q$, if I_1 and I_2 are adjacent at p and I_1 lies counterclockwise from I_2 along ∂Q. In this case, the direction of e is assigned from v_2 to v_1.

Given that $p \neq q$, the simple path γ_{pq} can be viewed as a simple path connecting two distinct points on the boundary of the circle and lying outside the circle. The path γ_{pq} can be classified into one of two types, depending on how it winds around the circle. Formally, γ_{pq} is homotopic to a directed path in $\mathbb{R}^2 \setminus \mathsf{int}(Q)$ that winds around ∂Q in either counterclockwise or clockwise direction. Note that it cannot wind around the boundary more than once since γ_{pq} is simple. We assign the direction of e from v_1 to v_2 if the path is of the counterclockwise type, and from v_2 to v_1, otherwise. Figure 5(b) illustrates γ_{pq} for both cases where $p = q$ and $p \neq q$. The same rule is applied to assign directions to all other edges in G_i.

In summary, we construct the directed graph G_i for $i \in [m]$ to represent how the disjoint components of R_i are to be connected into a single piece within Q. Each vertex corresponds to an interval in $\mathcal{I}_i^+$. Each directed edge $e = (v_1, v_2)$ of G_i represents a link instruction (I_1, I_2, k), but k is not specified yet. See Figure 5(c) for an illustration of the directed graph G_i that defines three link instructions, where P_i and Q are as shown in Figure 5(a).

4.4 Layer assignments for link instructions

For $i \in [m]$, let G_i be the directed graph for $R_i = \{C_1, C_2, \ldots, C_t\}$. Each edge in G_i represents a link instruction that reallocates layer segments within a specific layer. The goal is to assign layers to link instructions so that no redundant reallocation of layer segments is allowed and all elements in R_i are eventually merged into a single piece for every $i = 1, \ldots, m$.

The underlying graph of a digraph is its undirected version, obtained by ignoring the directions of all edges. Consider a connected component T of the underlying graph of G_i. We assume that T contains more than one vertex, as a component of size 1 does not indicate any link instruction. Each vertex of T corresponds to a circular interval in $\mathcal{I}_i^+$. Let $\mathcal{I}_T$ be the subset of $\mathcal{I}_i^+$ such that $\mathcal{I}_T = \{I \in \mathcal{I}_i^+ \mid I \text{ corresponds to a vertex in } T\}$.

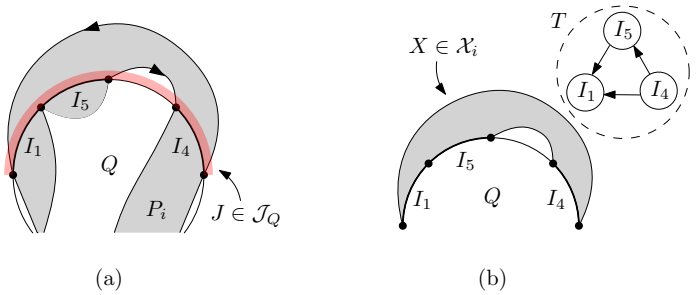


Figure 6 (a) The combine step merges I_1, I_5, and I_4 to a circular interval $J \in \mathcal{J}_Q$. (b) The interval J is derived from a connected component T of the underlying graph of G_i. This corresponds to a connected region $X \in \mathcal{X}_i$ that is bounded by the circular intervals associated with T.

We iteratively perform a combine step to merge all circular intervals in $\mathcal{I}_T$ into a single circular interval on ∂Q. The process begins with an initial set $S = \{u\}$ and a circular interval J_S, where u is any vertex of T and J_S is the circular interval in $\mathcal{I}_T$ corresponding to u. At each step, any vertex $v \in T \setminus S$ is chosen if there exists an edge between v and some vertex $w \in S$. Let $I_v \in \mathcal{I}_T$ be the circular interval corresponding to v. We add v to S and update J_S to be the smallest circular interval on ∂Q that spans from I_v to J_S in counterclockwise direction if the edge is directed from v to w, or in clockwise direction otherwise. Once $S = T$, the process returns a single circular interval on ∂Q, denoted by J_T, corresponding to the connected component T of G_i. See Figure 6(a) for an illustration of this process.

$$\mathcal{J}_Q \coloneqq \{J_T \mid T \text{ is a connected component of } G_i \text{ with } |T| > 1, \text{ for } i = 1, 2, \ldots, m\}.$$

Throughout, we use I to denote intervals in $\mathcal{I}_Q$ and J for those in $\mathcal{J}_Q$ to emphasize their distinct roles. Each $J \in \mathcal{J}_Q$ is non-degenerate. For any $J_1, J_2 \in \mathcal{J}_Q$ with $\mathsf{int}(J_1) \cap \mathsf{int}(J_2) \neq \emptyset$, there exists a proper containment between J_1 and J_2. To prove this, we present a lemma relating connectivity in the underlying graph of G_i to the existence of a path contained in a connected component of $P_i \setminus \mathsf{int}(Q)$, as illustrated in Figure 6(b).

▶ **Lemma 6.** *The vertices corresponding to I_1 and I_2 in $\mathcal{I}_i^+$ are connected in the underlying graph of G_i if and only if there is a simple path connecting $p \in I_1$ to $q \in I_2$ contained in $P_i \setminus \mathsf{int}(Q)$.*

Let $\mathcal{X}_i$ be the set of connected components in $P_i \setminus \mathsf{int}(Q)$. Then each element in $\mathcal{X}_i$ is a closed connected set. Moreover, it is a (weakly) simple polygon.

For $p, q \in \partial Q$, let $\Lambda(p, q)$ denote the set of all simple paths from p to q contained in $P_i \setminus \mathsf{int}(Q)$. For $I_1, I_2 \in \mathcal{I}_i^+$, we define $\Lambda(I_1, I_2) = \bigcup_{p \in I_1, q \in I_2} \Lambda(p, q)$. Lemma 6 guarantees the existence of a path between I_1 and I_2 whenever their corresponding vertices are connected in the underlying graph of G_i. Furthermore, all such paths share a consistent topological behavior, such as winding around ∂Q in the same direction.

▶ **Corollary 7.** *Let $I_1, I_2 \in \mathcal{I}_i^+$ with $I_1 \neq I_2$. If $\Lambda(I_1, I_2) \neq \emptyset$, then all paths in $\Lambda(I_1, I_2)$ wind around ∂Q in the same direction, and there is a unique component $X \in \mathcal{X}_i$ that contains all such paths.*

Using Lemma 6 and Corollary 7, we prove a proper containment relation.

▶ **Lemma 8.** *For any distinct $J_1, J_2 \in \mathcal{J}_Q$ with $\mathsf{int}(J_1) \cap \mathsf{int}(J_2) \neq \emptyset$, $J_1 \subsetneq J_2$ or $J_2 \subsetneq J_1$.*

By Lemma 8, we construct the circular interval graph $G_{\mathcal{J}}$, where each node corresponds to a circular interval in $\mathcal{J}_Q$, and a directed edge from J_1 to J_2 is added if $J_1 \subsetneq J_2$ for $J_1, J_2 \in \mathcal{J}_Q$. Then, $G_{\mathcal{J}}$ is acyclic as each edge corresponds to a strict containment.

Transitive reduction of $G_{\mathcal{J}}$. The graph $G_{\mathcal{J}}$ represents the transitive closure of the proper containment relation. That is, for $J_1, J_2 \in \mathcal{J}_Q$, there is a directed edge from J_1 to J_2 in $G_{\mathcal{J}}$ if there exists some $J' \in \mathcal{J}_Q$ such that (J_1, J') and (J', J_2) are directed edges in $G_{\mathcal{J}}$. A *transitive reduction* of $G_{\mathcal{J}}$ is a directed graph on the same vertex set with the minimum number of edges that preserves all reachability relations of $G_{\mathcal{J}}$. Since the transitive reduction of a DAG is unique [8], we denote the transitive reduction of $G_{\mathcal{J}}$ by $G_{\mathcal{J}}^{\mathrm{tr}}$. Let $UG_{\mathcal{J}}^{\mathrm{tr}}$ denote the underlying graph of $G_{\mathcal{J}}^{\mathrm{tr}}$.

▶ **Lemma 9.** *Any vertex of $G_{\mathcal{J}}^{\mathrm{tr}}$ has out-degree at most one, and $UG_{\mathcal{J}}^{\mathrm{tr}}$ is acyclic.*

By Lemma 9, $UG_{\mathcal{J}}^{\mathrm{tr}}$ is a forest. Each directed tree T of $G_{\mathcal{J}}^{\mathrm{tr}}$ has total out-degrees $|T| - 1$, implying that exactly one vertex has out-degree zero and all others have out-degree one. This structure corresponds to an *in-branching* tree [8].

Layer assignment of link instructions. In each in-branching tree of $G^{\mathrm{tr}}_{\mathcal{J}}$, the unique vertex with out-degree zero is called the root. The level of a vertex is defined as the number of edges on the path from the root to that vertex plus one, where the level of the root is one. The level of each interval $J \in \mathcal{J}_Q$ is defined to be the level of the corresponding vertex in $G^{\mathrm{tr}}_{\mathcal{J}}$.

Recall that each circular interval $J \in \mathcal{J}_Q$ is formed by merging intervals in $\mathcal{I}^+_i$ for some $i \in [m]$. These intervals correspond to the vertices of a connected component T of G_i, where each edge of T encodes a link instruction that merges two elements in R_i. Let $\texttt{INST}(J)$ denote the set of link instructions corresponding to the edges of T. We assign the k-th layer to every link instruction in $\texttt{INST}(J)$ if J is at level k in $G^{\mathrm{tr}}_{\mathcal{J}}$. Note that the maximum level in $G^{\mathrm{tr}}_{\mathcal{J}}$ is at most $\lfloor |\mathcal{I}^+_Q|/2 \rfloor$ if each connected component of G_i has size two for all $i = 1, 2, \ldots, m$. Thus, we set the number of layers in the corridor $\mathcal{A}$ to this maximum level: $h = \lfloor |\mathcal{I}^+_Q|/2 \rfloor$.

For each $J \in \mathcal{J}_Q$, all link instructions in $\texttt{INST}(J)$ reallocate layer segments within a single layer L_k to a common piece P_i, where $i \in [m]$ and $k \in [h]$ are determined by J. Let $I_a, I_b \in \mathcal{I}^+_i$ such that J is the smallest circular interval that spans from I_a to I_b in counterclockwise order. Let $Z_a, Z_b \subseteq L_k$ be the layer segments corresponding to I_a and I_b, respectively. Applying $\texttt{INST}(J)$ is equivalent to reallocating every $Z \in L_k[Z_a, Z_b]$ to P_i.

In summary, $\texttt{INST}_Q = \bigcup_{J \in \mathcal{J}_Q} \texttt{INST}(J)$ defines the set of all link instructions for reconfiguration. Applying $\texttt{INST}_Q$ to $\Pi[Q]$ yields a partition $Q = Q^*_1 \cup \cdots \cup Q^*_m$, where each Q^*_i is a region assigned to P_i within Q. When Q is $\overline{W}$-convex with respect to P, each Q^*_i is connected and satisfies both constraints W and U.

5 Analysis of reconfigured non-guillotine partitions

Assuming that the link instructions in $\texttt{INST}_Q$ have been applied to $\Pi[Q]$ in an arbitrary order, we verify the following statements in order to prove Theorem 3.

(1) For each $i = 1, \ldots, m$, the region allocated to P_i within Q forms a single connected piece.

(2) The reconfiguration of $\Pi[Q]$ is a solution to the problem $\mathsf{wPartition}(Q, W, U)$ if Q is $\overline{W}$-convex with respect to P.

It follows from statements (1) and (2) that the reconfigured partition of Q satisfies the constraints W and U while ensuring that the number of pieces does not exceed m.

5.1 Connectivity in reconfigured partitions

For $i \in [m]$, let $\texttt{INST}_i$ denote the subset of $\texttt{INST}_Q$ consisting of instructions of the form (I_1, I_2, k), where $I_1, I_2 \in \mathcal{I}^+_i$ and $k \in [h]$. Then, $\texttt{INST}_Q = \bigcup_{i \in [m]} \texttt{INST}_i$. We first show that the link instructions in $\texttt{INST}_i$ merge all elements of R_i into a single connected piece. We then verify that applying the link instructions in $\texttt{INST}_Q \setminus \texttt{INST}_i$ does not disconnect the merged piece. Furthermore, the resulting reconfigured partition of Q is invariant under the order in which the link instructions in $\texttt{INST}_Q$ are applied to $\Pi[Q]$.

Merging R_i via $\texttt{INST}_i$. We show that any two pieces $C_j, C_{j'} \in R_i$ are connected by some link instructions in $\texttt{INST}_i$. During the traversal on ∂P_i, each continuous path of $\partial C_j \cap \mathsf{int}(Q)$ is encountered at least once for every $j \in [t]$, and we denote the sequence of these paths in the order they are visited as $(\gamma_1, \gamma_2, \ldots, \gamma_l)$, where $\gamma_1 = \gamma_l$. Then, there exist indices $k, k' \in [l]$ such that $\gamma_k \subseteq \partial C_j \cap \mathsf{int}(Q)$ and $\gamma_{k'} \subseteq \partial C_{j'} \cap \mathsf{int}(Q)$. Since it is a cyclic sequence with $\gamma_1 = \gamma_l$, we assume without loss of generality that $k < k'$.

Recall that an edge of G_i is added whenever two consecutive paths in the sequence are derived from distinct elements in R_i. This indicates that a transition between them occurs during the traversal. The link instruction associated with this edge merges the corresponding

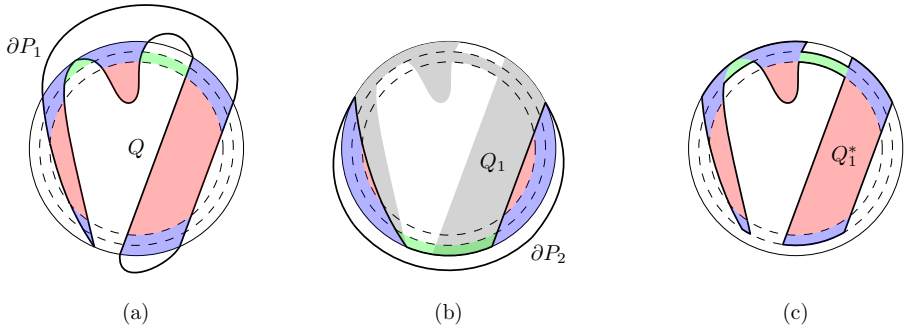


Figure 7 The corridor of Q has two layers, with $|R_1| = 3$ and $|R_2| = 2$. (a) Red regions represent $\{\text{core}(C) \mid \text{for } C \in R_1\}$, while blue and green regions indicate layer segments in $\mathcal{Z}_1^{\text{in}}$ and $\mathcal{Z}_1^{\text{out}}$, respectively. (b) Q_1 is formed by merging R_1 according to $\texttt{INST}_1$, and some layer segments in $\mathcal{Z}_1^{\text{in}}$ are reallocated by $\texttt{INST}_2$. (c) Q_1^* is obtained by applying $\texttt{INST}_Q \setminus \texttt{INST}_1$ to Q_1.

elements in R_i. The subsequence $(\gamma_k, \gamma_{k+1}, \ldots, \gamma_{k'})$ contains multiple transitions between distinct elements in R_i, starting from C_j and eventually reaching $C_{j'}$. Applying all link instructions associated with the transitions in $(\gamma_k, \ldots, \gamma_{k'})$ results in C_j and $C_{j'}$ being merged into a single connected piece.

Let Q_i denote the subregion of Q resulting from merging elements of R_i via $\texttt{INST}_i$, with no other instructions applied. Then, Q_i consists of all elements in R_i and layer segments reallocated from other pieces: $Q_i = \left(\bigcup_{j\in[t]} C_j\right) \cup \mathcal{Z}_i^{\text{out}}$, where $\mathcal{Z}_i^{\text{out}}$ is the set of layer segments reallocated to P_i by $\texttt{INST}_i$.

The region $\mathbf{L} = L_1 \cup \cdots \cup L_h$ denotes the union of all layers in Q. For each $C_j \in R_i$, $\mathbf{L} \cap C_j$ consists of layer segments within C_j. By construction of layers, $C_j \setminus \mathbf{L}$ is connected and non-empty. We refer to this region as a *core* of C_j, denoted by $\text{core}(C_j) = C_j \setminus \mathbf{L}$. Each C_j consists of its core together with the layer segments it contains. Let $\mathcal{Z}_i^{\text{in}}$ denote the set of all layer segments in $\mathbf{L} \cap C_j$ over all $C_j \in R_i$. Therefore, for $i \in [m]$, we have $Q_i = \left(\bigcup_{j\in[t]} \text{core}(C_j)\right) \cup \mathcal{Z}_i^{\text{in}} \cup \mathcal{Z}_i^{\text{out}}$. Figure 7(a) illustrates the parts of Q_i: cores and layer segments. The shapes are drawn schematically to reflect the topological structure, rather than an exact polygonal description.

We now turn to the link instructions in $\texttt{INST}_Q \setminus \texttt{INST}_i$ that are applied to Q_i. Let Q_i^* be a subregion of Q_i that is obtained by applying all link instructions in $\texttt{INST}_Q \setminus \texttt{INST}_i$ to Q_i. Note that any part of Q_i reallocated by $\texttt{INST}_Q \setminus \texttt{INST}_i$ lies within $\mathcal{Z}_i^{\text{in}} \cup \mathcal{Z}_i^{\text{out}}$, and thus the cores remain unchanged. We show that Q_i^* is well-defined, meaning that Q_i^* is invariant under the order in which the instructions in $\texttt{INST}_Q$ are applied.

▶ **Lemma 10.** *Let $(I_1, I_2, k) \in \texttt{INST}_i$ be a link instruction, and let Z_1 and Z_2 be the layer segments in L_k corresponding to I_1 and I_2, respectively. Then, the layer segments $L_k[Z_1, Z_2]$ remain assigned to P_i under any link instruction in $\texttt{INST}_Q \setminus \texttt{INST}_i$.*

Lemma 10 implies that no layer segment is reassigned by more than one link instruction. As a consequence, we obtain the following corollary, which states that Q_i^* is well-defined.

▶ **Corollary 11.** *In the reconfiguration of $\Pi[Q]$, each layer segment in Q is reallocated at most once.*

Corollary 11 ensures that each layer segment may be reallocated to a different piece at most once, and no chains of reallocations such as $P_{i_1} \to P_{i_2} \to P_{i_3}$ with $i_1 \neq i_2$ and $i_2 \neq i_3$ occur.

This lemma further implies that layer segments in $\mathcal{Z}_i^{\text{out}}$ are preserved, while only those in $\mathcal{Z}_i^{\text{in}}$ are reallocated by $\texttt{INST}_Q \setminus \texttt{INST}_i$. Let $\mathcal{Z}_i^{\text{in}*}$ be the subset of $\mathcal{Z}_i^{\text{in}}$ that consists of layer segments preserved under $\texttt{INST}_Q \setminus \texttt{INST}_i$. Then, Q_i^* is the subpolygon that is obtained from Q_i by removing those layer segments in $\mathcal{Z}_i^{\text{in}} \setminus \mathcal{Z}_i^{\text{in}*}$. Then $Q_i^* = \left(\bigcup_{j\in[t]} \text{core}(C_j)\right) \cup \mathcal{Z}_i^{\text{in}*} \cup \mathcal{Z}_i^{\text{out}}$, where $\mathcal{Z}_i^{\text{in}*} \subseteq \mathcal{Z}_i^{\text{in}}$. Figure 7(b–c) illustrates the construction of Q_i^*, in which only layer segments in $\mathcal{Z}_i^{\text{in}}$ are reallocated by $\texttt{INST}_Q \setminus \texttt{INST}_i$.

Path-Connectivity of Q_i^*. To prove that Q_i^* forms a connected piece, it suffices to verify two types of path-connectivity among its constituent parts, which must be preserved during the reallocation induced by $\texttt{INST}_Q \setminus \texttt{INST}_i$.

- Each layer segment in $\mathcal{Z}_i^{\text{in}*} \cup \mathcal{Z}_i^{\text{out}}$ is path-connected to some $\text{core}(C_j)$ within Q_i^* for $C_j \in R_i$.
- The cores $\{\text{core}(C_j) \mid C_j \in R_i\}$ are mutually path-connected within Q_i^*.

Here, two sets $A, B \subseteq X$ are said to be path-connected within a region X if there exists a path in X joining some $a \in A$ and $b \in B$.

▶ **Lemma 12.** *Each layer segment in $\mathcal{Z}_i^{\text{in}*} \cup \mathcal{Z}_i^{\text{out}}$ is path-connected within Q_i^* to the core of some $C_j \in R_i$.*

By Lemma 12, every layer segment in $\mathcal{Z}_i^{\text{in}*} \cup \mathcal{Z}_i^{\text{out}}$ has a path to some core within Q_i^*. It remains to show that $\text{core}(C_1), \ldots, \text{core}(C_t)$ are mutually path-connected within Q_i^*.

▶ **Lemma 13.** *All cores of elements in R_i are mutually path-connected within Q_i^*.*

By Lemmas 12 and 13, each Q_i^* is connected. Thus, applying $\texttt{INST}_Q$ to $\Pi[Q]$ yields the partition $\Pi^*[Q] = \{Q_1^*, Q_2^*, \ldots, Q_m^*\}$, where each Q_i^* is a connected subregion of Q.

Remarks. Applying link instructions in $\texttt{INST}_Q$ may induce holes within merged pieces in the reconfigured partition of Q. When Q_i^* contains holes, reallocating them to P_i does not increase $\omega_{\mathbf{v}}(Q_i^*)$ for every $\mathbf{v} \in \mathbb{S}^+$. Therefore, each Q_i^* can be regarded as a simple polygon.

5.2 Feasibility of reconfigured partitions

We first observe that the reconfigured partition $\Pi^*[Q] = \{Q_1^*, Q_2^*, \ldots, Q_m^*\}$ remains as a valid partition of Q. By construction, the cut constraint U is also preserved: each layer segment is bounded by the boundaries of layers and the cuts from $\Pi[Q]$, all aligned with directions in U. It remains to check the unit-width constraint W.

Since $\Pi = \{P_1, \ldots, P_m\}$ is a solution to $\mathsf{wPartition}(P, W, U)$, there exists a vector $\mathbf{v_i} \in W$ such that $\omega_{\mathbf{v_i}}(P_i) \leq 1$. In other words, there exists a unit strip H_i with normal vector $\mathbf{v_i}$ that contains P_i. If every layer segment reallocated by $\texttt{INST}_i$ is contained within H_i, then Q_i^* also satisfies unit-width constraint W.

Assume that $\lambda = (I_1, I_2, k')$ is a link instruction associated with a directed edge from v_1 to v_2 in G_i for $I_1, I_2 \in \mathcal{I}_i^+$ and $k' \in [h]$. Let $j, j' \in [t]$ be two distinct indices such that $I_1 \in \mathcal{I}_{ij}^+$ and $I_2 \in \mathcal{I}_{ij'}^+$. Recall that the edge between v_1 and v_2 is added to G_i if and only if there is a transition between ∂C_j and $\partial C_{j'}$ during the counterclockwise traversal on ∂P_i.

For each $k \in [h]$, let Z_1^k and Z_2^k denote the layer segments in L_k corresponding to I_1 and I_2, respectively. The link instruction λ reallocates the layer segments in $L_{k'}(Z_1^{k'}, Z_2^{k'})$ to P_i, where $Z_1^{k'}$ and $Z_2^{k'}$ are already assigned to P_i in $\Pi[Q]$. We define $\mathbf{Z}_\lambda \coloneqq \bigcup_{k\in[h]} \bigcup_{Z\in\mathcal{Z}^k} Z$, where $\mathcal{Z}^k \coloneqq L_k(Z_1^k, Z_2^k)$. It suffices to show that $\mathbf{Z}_\lambda \subseteq H_i$. If this inclusion holds, every layer segment reallocated by λ lies within H_i, and thus the unit-width constraint W is preserved.

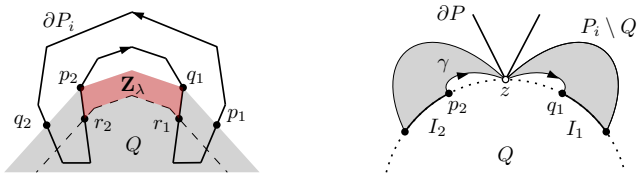


Figure 8 (a) The link instruction $\lambda = (I_1, I_2, k)$ is derived from the traversal of ∂P_i from p_1 to q_2 or from p_2 to q_1, where $I_1 = \partial Q[p_1, q_1]$ and $I_2 = \partial Q[p_2, q_2]$. $\partial \mathbf{Z}_\lambda$ consists of $\partial Q[q_1, p_2], \partial Q^\phi[r_1, r_2], \overline{q_1 r_1}$, and $\overline{p_2 r_2}$. (b) The path γ from p_2 to q_1 must pass through z when ∂P intersects ∂Q at z.

Decomposition of $\partial \mathbf{Z}_\lambda$. The region $\mathbf{Z}_\lambda$ is connected and bounded by four parts: two continuous portions of the inner and outer boundaries of $\mathcal{A}$, and two subsegments of edges from C_j and $C_{j'}$. The outer boundary of $\mathcal{A}$ is ∂Q and its inner boundary is ∂Q^ϕ, where Q^ϕ is the inner ϕ-offset polygon of Q.

For the sake of clarity, we introduce interval notation to represent subpaths of ∂Q and ∂Q^ϕ. For any two points $x, y \in \partial Q$, we denote by $\partial Q[x, y]$ the portion of ∂Q from x to y in counterclockwise order, including both endpoints. Let $\partial Q(x, y] = \partial Q[x, y] \setminus \{x\}$, $\partial Q[x, y) = \partial Q[x, y] \setminus \{y\}$, and $\partial Q(x, y) = \partial Q[x, y] \setminus \{x, y\}$. Similarly, we define $\partial Q^\phi[x, y], \partial Q^\phi(x, y], \partial Q^\phi[x, y)$, and $\partial Q^\phi(x, y)$ as portions of ∂Q^ϕ.

Since I_1 and I_2 are non-degenerate intervals on ∂Q, let $I_1 = \partial Q[p_1, q_1]$ and $I_2 = \partial Q[p_2, q_2]$ for some points $p_1, q_1, p_2, q_2 \in \partial Q$ with $p_1 \neq q_1$ and $p_2 \neq q_2$. The portion of $\partial \mathbf{Z}_\lambda$ contained in ∂Q is $\partial Q[q_1, p_2]$. Let $r_1, r_2 \in \partial Q^\phi$ be the endpoints of the portion of $\partial \mathbf{Z}_\lambda$ lying on ∂Q^ϕ, which we denote by $\partial Q^\phi[r_1, r_2]$. Finally, the parts of $\partial \mathbf{Z}_\lambda$ along the edges of C_j and $C_{j'}$ correspond to the segments $\overline{q_1 r_1}$ and $\overline{p_2 r_2}$, respectively. Thus, $\partial \mathbf{Z}_\lambda$ decomposes into the four parts $\partial Q[q_1, p_2], \partial Q^\phi[r_1, r_2], \overline{q_1 r_1}$, and $\overline{p_2 r_2}$. This decomposition is shown in Figure 8(a).

Since H_i is convex, $\mathbf{Z}_\lambda \subseteq H_i$ if and only if all parts of $\partial \mathbf{Z}_\lambda$ are contained in H_i. Note that $\overline{q_1 r_1}$ and $\overline{p_2 r_2}$ lie in H_i, as both C_j and $C_{j'}$ are contained in P_i. To prove $\mathbf{Z}_\lambda \subseteq H_i$, it remains to show that $\partial Q[q_1, p_2]$ and $\partial Q^\phi[r_1, r_2]$ are contained in H_i.

Containment of $\partial Q[q_1, p_2]$ in $\mathsf{int}(P)$. During the transition between ∂C_j and $\partial C_{j'}$ in the counterclockwise traversal on ∂P_i, it follows a simple path, denoted by γ, which lies outside $\mathsf{int}(Q)$. The path γ exits and re-enters $\mathsf{int}(Q)$ through the endpoints of I_1 and I_2.

Let v_1 and v_2 denote the vertices in G_i corresponding to I_1 and I_2, respectively. The link instruction $\lambda = (I_1, I_2, k')$ is derived from the directed edge (v_1, v_2) in G_i. The direction of the edge is determined by how γ winds around ∂Q (either clockwise or counterclockwise) and whether the path exits $\mathsf{int}(Q)$ through the endpoint of I_1 or that of I_2. If γ winds around ∂Q counterclockwise, it exits $\mathsf{int}(Q)$ from p_1 and re-enters at q_2. Otherwise, it exits $\mathsf{int}(Q)$ from p_2 and re-enters at q_1. Figure 8(a) illustrates both cases.

Up to this point, we consider both clockwise- and counterclockwise-type instructions. However, counterclockwise ones can be omitted in the reconfiguration. Assume that γ winds around ∂Q counterclockwise. By Corollary 7, there exists a unique component $X \in \mathcal{X}_i$ whose boundary contains I_1, I_2, and the path γ. Since X is a weakly simple polygon, we can traverse ∂X in counterclockwise order. This traversal encounters a sequence of circular intervals in $\mathcal{I}_i^+$, and, from the construction of G_i, each consecutive pair of intervals in this sequence corresponds to an edge in G_i whose direction is determined by whether the subpath of ∂X between the intervals winds around ∂Q clockwise or counterclockwise.

Since the traversal of ∂X follows the path γ from I_1 to I_2 that winds around ∂Q counterclockwise, the other subpath of ∂X runs from I_2 back to I_1, and winds around ∂Q clockwise. This path encounters a sequence of intervals in $\mathcal{I}_i^+$ starting from I_2 to I_1, where each consecutive pair of intervals induces a clockwise-type edge in G_i. In Figure 6(b), the counterclockwise-type path from I_4 to I_1 corresponds to the sequence of clockwise-type paths $I_1 \to I_5$ and $I_5 \to I_4$. Thus, link instructions associated with counterclockwise-type edges can be omitted without affecting the resulting partition $\Pi^*[Q]$.

Without loss of generality, we restrict our analysis to clockwise type instructions. In this case, γ is a path from p_2 to q_1. The path γ can be continuously deformed into a path $\tilde{\gamma} = \partial Q[q_1, p_2]$ on ∂Q while preserving its endpoints. As illustrated in Figure 8(b), if ∂P intersects Q at a point $z \in \partial Q[q_1, p_2]$, then γ must pass through z. This implies that ∂P_i intersects itself at z, contradicting that P_i is a simple polygon.

▶ **Lemma 14.** *All points on $\partial Q[q_1, p_2]$ lie within the interior of P.*

Note that a point $p \in \mathsf{int}(P)$ if and only if there exists a sufficiently small ball $B(p)$ centered at p such that $B(p) \subseteq \mathsf{int}(P)$. We slightly extend the polygonal chain $\partial Q[q_1, p_2]$ beyond its endpoints and denote the resulting chain by $\partial Q[q_1^-, p_2^+]$ for $q_1^- \in \partial Q(p_1, q_1)$ and $p_2^+ \in \partial Q(p_2, q_2)$. Since $\partial Q[q_1, p_2] \subseteq \mathsf{int}(P)$ by Lemma 14, $\partial Q[q_1^-, p_2^+]$ also lies in $\mathsf{int}(P)$. Furthermore, as I_1 and I_2 are non-degenerate, the slight extension guarantees that $\partial Q[q_1^-, p_2^+] \subseteq \partial Q[p_1, q_2]$.

Turning points on $\partial Q[q_1^-, p_2^+]$. Without loss of generality, assume H_i is a vertical strip, meaning that its boundary consists of two vertical lines. A portion of ∂Q is a polygonal chain, consisting of a sequence of line segments. As we traverse a polygonal chain from one endpoint to the other, these line segments are encountered sequentially. A point $z \in \partial Q(q_1^-, p_2^+)$ is called a *turning point* of $\partial Q[q_1^-, p_2^+]$ if the traversal changes its horizontal direction (from leftward to rightward or vice versa) at z. Since q_1^- and p_2^+ lie sufficiently close to q_1 and p_2 along ∂Q, every turning must occur on $\partial Q[q_1, p_2]$.

When $\partial Q[q_1^-, p_2^+]$ contains vertical edges, the above definition of turning points is not sufficient. Consider a path that initially moves in the positive (or negative) x-direction, then follows a vertical segment, and subsequently moves in the negative (or positive) x-direction. We define the lowest point on the vertical segment as the unique turning point on that segment. See Figure 9(a).

Note that the number of turning points is invariant under the traversal direction; that is, it remains unchanged whether we traverse $\partial Q[q_1^-, p_2^+]$ from q_1^- to p_2^+ or in the reverse direction. As illustrated in Figure 9(b), if there are two turning points on $\partial Q[q_1^-, p_2^+]$, we can draw a vertical guillotine cut c in P such that $c \cap Q$ is disconnected, since all points on $\partial Q[q_1^-, p_2^+]$ lie within $\mathsf{int}(P)$. This implies that Q cannot be $\overline{W}$-convex with respect to P.

▶ **Lemma 15.** *If Q is $\overline{W}$-convex with respect to P, then the number of turning points on $\partial Q[q_1^-, p_2^+]$ is at most one.*

Since both $\partial Q[p_1, q_1]$ and $\partial Q[p_2, q_2]$ lie in P_i, $q_1^-, p_2^+ \in H_i$. If there are no turning points on $\partial Q[q_1^-, p_2^+]$, then the points with the largest and smallest x-coordinates along $\partial Q[q_1^-, p_2^+]$ appear at q_1^- and p_2^+. Since H_i is a vertical slab, all points on $\partial Q[q_1^-, p_2^+]$ lie in H_i.

Consider the case that $\partial Q[q_1^-, p_2^+]$ has a turning point at $z \in \partial Q[q_1, p_2]$, and it is unique by Lemma 15. Then, the point with the largest or smallest x-coordinate along $\partial Q[q_1^-, p_2^+]$ may appear at z. Without loss of generality, we assume that z is the point with the largest x-coordinate. The point with the smallest x-coordinate lies at $q_1^- \in H_i$ or $p_2^+ \in H_i$.

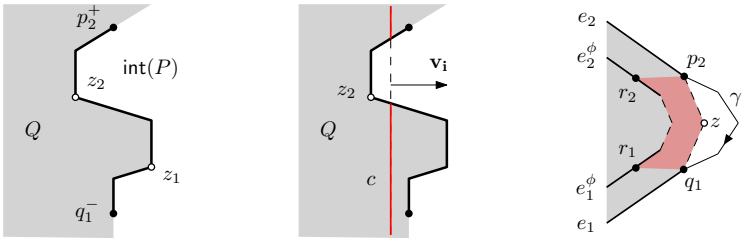


Figure 9 (a) The turning points on $\partial Q[q_1^-, p_2^+]$ are z_1 and z_2. (b) For the guillotine cut c along $x = x(z) + \varepsilon$, the intersection $c \cap Q$ consists of at least two maximal line segments. (c) γ contains a point with x-coordinate larger than that of the turning point z. The sequence of edges on $\partial Q^\phi[r_1, r_2]$ from e_1^ϕ to e_2^ϕ corresponds to a subsequence of those of $\partial Q[p_1, q_2]$.

Recall that the path γ follows from p_2 to q_1 outside $\mathsf{int}(Q)$ and is deformed into the path $\tilde{\gamma}$ that traverses $\partial Q[q_1, p_2]$. Note that z is a convex vertex of Q with locally largest x-coordinate, and γ encloses z from outside $\mathsf{int}(Q)$. It follows that γ must pass through a point with x-coordinate at least $x(z)$. Since $\gamma \subseteq P_i \subseteq H_i$, the x-coordinate of z is smaller than that of the right boundary of H_i. Thus, $\partial Q[q_1^-, p_2^+] \subseteq H_i$. See Figure 9(c).

Containment of $\mathbf{Z_\lambda}$ within H_i. Revisiting the boundary of $\mathbf{Z}_\lambda$, we have shown that three parts, $\overline{r_1 q_1}, \overline{r_2 p_2}$, and $\partial Q[q_1, p_2]$, are contained in H_i. The remaining part is $\partial Q^\phi[r_1, r_2]$ which is the portion of the inner ϕ-offset polygon of Q. By definition of the offset polygon, each edge of Q^ϕ is parallel to its corresponding edge in Q and the edges of Q^ϕ appear in the same cyclic order along its boundary as the edges of Q.

We traverse the chain $\partial Q[q_1^-, p_2^+]$ from q_1^- to p_2^+. Let e_1 and e_2 be the edges of Q that contain the first and last segments of this chain, respectively. Likewise, traversing $\partial Q^\phi[r_1, r_2]$ from r_1 to r_2 gives edges e_1^ϕ, e_2^ϕ of Q^ϕ incident to r_1 and r_2, respectively. We claim that the edges of Q corresponding to e_1^ϕ and e_2^ϕ appear along the traversal of $\partial Q[q_1^-, p_2^+]$.

The segments $\overline{r_1 q_1}$ and $\overline{r_2 p_2}$ lie on edges of C_j and $C_{j'}$, respectively. Since $\phi < \phi_{ij}$ and $\phi < \phi_{ij'}$, the edges of Q corresponding to e_1^ϕ and e_2^ϕ are incident to q_1 and p_2, respectively. Recall that we work on the extended chain $\partial Q[q_1^-, p_2^+]$, obtained by slightly extending $\partial Q[q_1, p_2]$. This guarantees that the corresponding edges of Q appear along $\partial Q[q_1^-, p_2^+]$. Consequently, the sequence of line segments forming $\partial Q^\phi[r_1, r_2]$ corresponds to a subsequence of those forming $\partial Q[q_1^-, p_2^+]$. See Figure 9(c) for an illustration of this correspondence. Thus, by Lemma 15, the number of turning points of $\partial Q^\phi[r_1, r_2]$ is also at most one.

Assuming that the largest x-coordinate of $\partial Q[q_1^-, p_2^+]$ occurs at its turning point, the largest x-coordinate of $\partial Q^\phi[r_1, r_2]$ is smaller than that of $\partial Q[q_1^-, p_2^+]$. The argument is symmetric when the smallest x-coordinate is attained at the turning point. It follows that $\partial Q^\phi[r_1, r_2]$ is also contained in H_i, which completes the proof that $\partial \mathbf{Z}_\lambda \subseteq H_i$. Hence, the reconfigured partition $\Pi^*[Q] = \{Q_1^*, \ldots, Q_m^*\}$ is a solution to the problem $\mathsf{wPartition}(Q, W, U)$ with at most m connected pieces; thus, by definition, $\mathsf{opt}(Q) \leq m = \mathsf{opt}(P)$.

6 Bang-type theorem for partitions of a convex body

We adapt the reconfiguration technique in Section 4 to prove Theorem 1. We then show that, when $\overline{W} \subseteq U$, an optimal partition of a convex polygon P is achieved by equally spaced parallel cuts, which can be computed in linear time.

Let K be a convex body in $\mathbb{R}^2$, and let $P_1 \cup P_2 \cup \cdots \cup P_m$ be its arbitrary partition. Note that each P_i is compact and possibly non-convex. Let $\mathrm{CH}(X)$ denote a convex hull of a set X in $\mathbb{R}^2$. A pocket of $\mathrm{CH}(X)$ is defined as a closure of a connected component of $\mathrm{CH}(X) \setminus X$. Each pocket is bounded by a subpath along ∂P_i and a unique line segment lying outside P_i. We refer to this segment as the *hull-edge* of the pocket.

To convexify P_i, we iteratively reallocate its pockets to P_i. However, such a reallocation may split other pieces P_j with $j \neq i$. Accordingly, we perform a reconfiguration step to merge such fragments into a single piece in K, ensuring that each piece remains connected.

This configuration mirrors Section 4, but with spatial roles reversed. Previously, we considered the restriction of a partition to a subpolygon Q, and reconnected the fragments of other pieces within Q. Here, we restrict the partition to the complement of a pocket, and consider the fragments of other pieces that lie outside the pocket. The circular intervals in the earlier setting now correspond to the intervals along the hull-edge of the pocket.

▶ **Lemma 16.** *Let $\{P_1, \ldots, P_m\}$ be a partition of a convex body $K \subseteq \mathbb{R}^2$. Then, there exists a convex partition $K = P_1^* \cup \cdots \cup P_m^*$, such that $\omega_{\mathbf{v}}(P_i) \geq \omega_{\mathbf{v}}(P_i^*)$ for all $\mathbf{v} \in \mathbb{S}^+$ and $i \in [m]$.*

By Lemma 16, we have a convex partition $K = P_1^* \cup \cdots \cup P_m^*$ such that $\omega_{\mathbf{v}}(P_i) \geq \omega_{\mathbf{v}}(P_i^*)$ for all $\mathbf{v} \in \mathbb{S}^+$ and $i \in [m]$. Given this convex partition, Akopyan [4] showed that $\sum_{i=1}^m r_K(P_i^*) \geq 1$, where $r_K(P_i^*) = \sup\{h \geq 0 \mid \exists t \in \mathbb{R}^2 \text{ such that } hK + t \subseteq P_i^*\}$. For any direction $\mathbf{v} \in \mathbb{S}^+$, we have $r_K(P_i^*) \leq \omega_{\mathbf{v}}(P_i^*)/\omega_{\mathbf{v}}(K)$. Thus, for any subset $W \subseteq \mathbb{S}^+$, $\sum_{i=1}^m \inf_{\mathbf{v}\in W} \frac{\omega_{\mathbf{v}}(P_i)}{\omega_{\mathbf{v}}(K)} \geq \sum_{i=1}^m \inf_{\mathbf{v}\in W} \frac{\omega_{\mathbf{v}}(P_i^*)}{\omega_{\mathbf{v}}(K)} \geq \sum_{i=1}^m r_K(P_i^*) \geq 1$.

Optimal partition for a convex polygon. Let P be a convex polygon with n vertices, and let $W, U \subseteq \mathbb{S}^+$ such that $\overline{W} \subseteq U$. Choose an arbitrary vector $\mathbf{u} \in W$; without loss of generality, assume that $\mathbf{u} = (1, 0)$. Let q be the leftmost vertex of P, and partition P by vertical lines along $x_i = x(q) + i$ for all $i = 1, \ldots, \lceil \omega_{\mathbf{u}}(P) \rceil - 1$. This partitions P into $\lceil \omega_{\mathbf{u}}(P) \rceil$ pieces, each of horizontal width at most 1, and it is a feasible solution to $\mathsf{wPartition}(P, W, U)$. Since $\mathbf{u}$ is chosen arbitrarily, $\mathsf{opt}(P) \leq \min_{\mathbf{v}\in W} \lceil \omega_{\mathbf{v}}(P) \rceil$.

Suppose, for the sake of contradiction, that the optimal partition has fewer than $\min_{\mathbf{v}\in W} \lceil \omega_{\mathbf{v}}(P) \rceil$ pieces. Let $P = P_1 \cup \cdots \cup P_m$ be an optimal partition for $\mathsf{wPartition}(P, W, U)$, with $m = \mathsf{opt}(P)$. By Theorem 1, we have $1 \leq \sum_{i=1}^m \inf_{\mathbf{v}\in W} (\omega_{\mathbf{v}}(P_i)/\omega_{\mathbf{v}}(P))$. Since each P_i satisfies unit-width constraint W, there exists $\mathbf{u}_i \in \mathbb{S}^+$ such that $\omega_{\mathbf{u}_i}(P_i) \leq 1$. Thus, $1 \leq \sum_{i=1}^m \inf_{\mathbf{v}\in W} (\omega_{\mathbf{v}}(P_i)/\omega_{\mathbf{v}}(P)) \leq \sum_{i=1}^m (\omega_{\mathbf{u}_i}(P_i)/\omega_{\mathbf{u}_i}(P)) \leq \sum_{i=1}^m (1/\omega_{\mathbf{u}_i}(P))$.

We analyze two cases depending on whether $\omega_{\mathbf{v}}(P)$ attains a minimum over W. If it does, we have $1 \leq \sum_{i=1}^m (1/\min_{\mathbf{v}\in W} \omega_{\mathbf{v}}(P))$, and $\min_{\mathbf{v}\in W} \omega_{\mathbf{v}}(P) \leq m$. As m is an integer, $\min_{\mathbf{v}\in W} \lceil \omega_{\mathbf{v}}(P) \rceil \leq m$. If no minimum is attained over W, $1 < \sum_{i=1}^m (1/\inf_{\mathbf{v}\in W} \omega_{\mathbf{v}}(P))$ and $\inf_{\mathbf{v}\in W} \omega_{\mathbf{v}}(P) < m$. Then $\min_{\mathbf{v}\in W} \lceil \omega_{\mathbf{v}}(P) \rceil = \inf_{\mathbf{v}\in W} \lceil \omega_{\mathbf{v}}(P) \rceil \leq m$. Both cases contradict our assumption, and thus $\mathsf{opt}(P) = \min_{\mathbf{v}\in W} \lceil \omega_{\mathbf{v}}(P) \rceil$.

Let $\mathbf{u} \in W$ be a vector minimizing $\lceil \omega_{\mathbf{u}}(P) \rceil$, computed in $O(n)$ time for $W = \mathbb{S}^+$ [18]. Assume that $\mathbf{u}$ is given. For the vertices of P given in counterclockwise order, such $m - 1$ parallel cuts can be computed in $O(\min\{n, m \log \frac{n}{m}\})$ time [12]. Since $x/(1+x) < \log(1+x) < x$ for all $x > 0$, $\min\{n, m \log \frac{n}{m}\} = \Theta(m \log(1 + \frac{n}{m}))$. For fixed n, $f(m) = m \log(1 + \frac{n}{m})$ starts at $\Theta(\log n)$ when $m = 1$ and increases monotonically, approaching $\Theta(n)$ as $m \to \infty$.

▶ **Corollary 17.** *Let P be a convex polygon with n vertices, and let $U, W \subseteq \mathbb{S}^+$ be sets of unit vectors such that $\overline{W} \subseteq U$. Then an optimal partition for the problem $\mathsf{wPartition}(P, W, U)$ is achieved by equally spaced parallel cuts orthogonal to $\mathbf{u} \in W$ that minimizes $\lceil \omega_{\mathbf{u}}(P) \rceil$. Given such a direction $\mathbf{u}$, the partition can be computed in $O(\omega_{\mathbf{u}}(P) \log(1 + \frac{n}{\omega_{\mathbf{u}}(P)}))$ time.*

Remarks. We work in the Real RAM model, which supports unit-cost arithmetic $(+, -, \times, \div)$ and comparisons on real numbers. Our algorithms perform integer rounding via comparisons.

References

1 Mikkel Abrahamsen and Nichlas Langhoff Rasmussen. Partitioning a polygon into small pieces. In *Proc. 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. Society for Industrial and Applied Mathematics, 2025.

2 Mikkel Abrahamsen and Jack Stade. Hardness of packing, covering and partitioning simple polygons with unit squares. In *Proc. 65th IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2024. doi:10.1109/FOCS61266.2024.00087.

3 Arseniy Akopyan and Roman Karasev. Kadets-type theorems for partitions of a convex body. *Discrete & Computational Geometry*, 48:766–776, 2012. doi:10.1007/S00454-012-9437-1.

4 Arseniy Akopyan, Roman Karasev, and Fedor Petrov. Bang's problem and symplectic invariants. *Journal of Symplectic Geometry*, 17(6):1579–1611, 2019.

5 Ralph Alexander. A problem about lines and ovals. *The American Mathematical Monthly*, 75(5):482–487, 1968.

6 Keith Ball. The plank problem for symmetric bodies. *Inventiones Mathematicae*, 104:535–543, 1991.

7 Thøger Bang. A solution of the "plank problem". *Proc. American Mathematical Society*, 2(6):990–993, 1951.

8 Jørgen Bang-Jensen and Gregory Z. Gutin. *Digraphs: Theory, Algorithms and Applications*. Springer Science & Business Media, 2008.

9 Andras Bezdek and Károly Bezdek. Conway's fried potato problem revisited. *Archiv der Mathematik*, 66(6):522–528, 1996.

10 Károly Bezdek. Plank theorems via successive inradii. In *Discrete Geometry and Algebraic Combinatorics, Contemporary Mathematics*, volume 625, pages 1–8. American Mathematical Society, 2013. URL: http://www.ams.org/books/conm/625/12489.

11 Maike Buchin and Leonie Selbach. Decomposing polygons into fat components. In *Proc. 33rd Canadian Conference on Computational Geometry (CCCG)*, pages 175–184, 2021.

12 Jaehoon Chung, Sang Won Bae, Chan-Su Shin, Sang Duk Yoon, and Hee-Kap Ahn. Approximating convex polygons by histogons. In *Proc. 34th Canadian Conference on Computational Geometry (CCCG)*, 2022.

13 Mirela Damian. Exact and approximation algorithms for computing optimal fat decompositions. *Computational Geometry: Theory and Applications*, 28(1):19–27, 2004. doi:10.1016/J.COMGEO.2004.01.004.

14 Mirela Damian and Sriram V. Pemmaraju. Computing optimal diameter-bounded polygon partitions. *Algorithmica*, 40(1):1–14, 2004. doi:10.1007/S00453-004-1092-3.

15 Mirela Damian-Iordache. *Shape constrained polygon decomposition and graph domination problems*. PhD thesis, University of Iowa, Iowa City, IA 52242, 2000.

16 Richard John Gardner. Relative width measures and the plank problem. *Pacific Journal of Mathematics*, 135(2):299–312, 1988.

17 J. W. S. Hearle. The structural mechanics of fibers. *Journal of Polymer Science Part C: Polymer Symposia*, 20:215–230, 1967.

18 Michael B. Houle and Godfried T. Toussaint. Computing the width of a set. In *Proc. 1st Annual Symposium on Computational Geometry (SCG)*, pages 1–7, 1985.

19 Henry F. Hunter. Some special cases of bang's inequality. *Proc. American Mathematical Society*, 117(3):819–821, 1993.

20 Joseph O'Rourke and Geetika Tewari. The structure of optimal partitions of orthogonal polygons into fat rectangles. *Computational Geometry*, 28(1):49–71, 2004. doi:10.1016/J.COMGEO.2004.01.007.

21 Gregory J. E. Rawlins. *Explorations in restricted orientation geometry*. PhD thesis, Indiana University, Department of Computer Science, Bloomington, 1987.

22 Chris Worman. Decomposing polygons into bounded diameter components. In *Proceedings of the 15th Canadian Conference on Computational Geometry (CCCG)*, pages 103–107, 2003.

Precoloring Extension with Demands on Paths

Arun Kumar Das
School of Computer and Information Sciences, University of Hyderabad, India

Michal Opler
Faculty of Information Technology, Czech Technical University in Prague, Czech Republic

Tomáš Valla
Faculty of Information Technology, Czech Technical University in Prague, Czech Republic

Abstract

Let G be a graph with a set of precolored vertices, and let us be given an integer distance parameter d and a set of integer *demands* $d_1, \ldots, d_c$. The *Distance Precoloring Extension with Demands* (DPED) problem is to compute a vertex c-coloring of G such that the following three conditions hold: (i) the resulting coloring respects the colors of the precolored vertices, (ii) the distance of two vertices of the same color is at least d, and (iii) the number of vertices colored by color i is exactly d_i. This problem is motivated by a program scheduling in commercial broadcast channels with constraints on content repetition and placement, which leads precisely to the DPED problem for paths.

In this paper, we study DPED on paths and present a polynomial time exact algorithm when precolored vertices are restricted to the two ends of the path and devise an approximation algorithm for DPED with an additive approximation factor polynomially bounded by d and the number of precolored vertices. Then, we prove that the *Distance Precoloring Extension* problem on paths, a less restrictive version of DPED without the demand constraints, and then DPED itself, is NP-complete. Motivated by this result, we further study the parameterized complexity of DPED on paths. We establish that the DPED problem on paths is $W[1]$-hard when parameterized by the number of colors and the distance. On the positive side, we devise a fixed parameter tractable (FPT) algorithm for DPED on paths when the number of colors, the distance, and the number of precolored vertices are considered as the parameters. Moreover, we prove that *Distance Precoloring Extension* is FPT parameterized by the distance. As a byproduct, we also obtain several results for the *Distance List Coloring* problem on paths.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Problems, reductions and completeness

Keywords and phrases precoloring extension, distance coloring, FPT, approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.23

Related Version *Full Version*: `https://arxiv.org/abs/2509.18936`

Funding This work was co-funded by the European Union under the project Robotics and advanced industrial production (reg. no. CZ.02.01.01/00/22_008/0004590).
Michal Opler: This work was supported by the Czech Science Foundation Grant no. 24-12046S.
Tomáš Valla: This work was supported by the Czech Science Foundation Grant no. 24-12046S.

Acknowledgements We would like to thank Václav Blažej, Dušan Knop, Josef Malík and Ondřej Suchý, with whom we held preliminary discussions and obtained several observations relevant to this research.

1 Introduction

Vertex coloring of graphs is one of the most studied and fundamental problems in structural and algorithmic graph theory. A *k-coloring* of a given graph partitions its vertices into k subsets (color classes) such that there is no edge between two members in the same partition. Graph coloring problems [19] have a rich research history [10, 22, 33, 7], comprising

© Arun Kumar Das, Michal Opler, and Tomáš Valla;
licensed under Creative Commons License CC-BY 4.0

numerous variants and results, due to their significant applications in scheduling [4], resource allocation [26], compiler design [30], computational biology [20], network analysis [3], and geography [14]. We study a particular variant of the vertex coloring problem with three additional constraints. Some of the vertices are already given precolored on input, and the sought coloring must respect these colors. Additionally, we are given required sizes of all color classes (called demands), and finally, we require any two vertices of the same color to lie at a distance of at least d from each other. A vertex coloring that satisfies the last condition is called a *d-distance coloring*. Hence, classical vertex colorings are exactly 1-distance colorings. Alternatively, d-distance colorings of a graph G correspond to classical vertex colorings of the dth power of G.

Formally, the problem we consider is stated as follows.

Distance Precoloring Extension with Demands (DPED)
Input: A graph $G = (V, E)$ on n vertices, a set of colors C, a non-negative integer d, a partial pre-coloring $\gamma' \colon A \to C$ for some $A \subseteq V$, and a demand function $\eta \colon C \to [n]$.
Question: Is there a d-distance coloring $\gamma \colon V \to C$ that
- γ extends γ' (i.e., $\gamma(v) = \gamma'(v)$ for every $v \in A$), and
- for every $c \in C$, the number of vertices newly colored by color c is exactly $\eta(c)$, i.e., $|\{v \in V \setminus A \mid \gamma(v) = c\}| = \eta(c)$.

We address the problem for paths and disjoint unions of paths. Our research is motivated by a real-world scheduling problem of a television broadcasting company: A commercial block of a daily broadcast consists of n slots (n vertices of a path), where some slots have already been allocated the type of commercial (precolored vertices). The commercials are already paid for (the demands are given), and no two commercials of the same type may be allocated close together. The task is thus to find an assignment of commercials to slots (color the vertices) such that no two commercials of the same type are too close (vertices being too close on the path must be colored by distinct colors), and all of the commercials (demands) are used. This original setting leads to a lot of interesting generalizations.

A notable amount of previous research has been done in the direction of the coloring of graphs and most of the problems are proved to be NP-hard depending on the type of the input. There are numerous complexity results for different variants of the coloring problems, for example, in terms of parameters like treewidth [13], distance to cluster and co-cluster [15], number of colors and maximum degree [9], clique modulator [17], or vertex cover [18]. Biró et al. [5] introduced the precoloring extension problem for interval graphs. They proved that the problem is polynomial time solvable when each color is used at most once for precoloring, and NP-complete when they are used twice or more. They also extended the results for graphs with bounded treewidth. They mentioned the problem as a variant of the List Coloring problem [11]. In this problem, each vertex is assigned with a list of colors, and they must be colored with a color from the assigned list. The graph is called *L-list colorable* if there exists a valid vertex coloring which chooses every color from the assigned lists L, and it is called *k-choosable* if this is possible for any assignment of lists of size at least k. List coloring has been studied on trees [6], planar graphs [32], and many other graph classes [29].

Yet another studied variant of coloring, the Equitable Coloring problem [25], requires the difference between the total number of vertices colored with two different colors to be at most one. Hence Coloring with Demands can be seen as a direct generalization of this notion. In practice, it is common to combine multiple constraints for coloring problems in order to address various applications, e.g., Pelsmajer [27] studied the problem of equitable list coloring. In the Distance Coloring problem, monochromatic vertices cannot occur

within a given distance from each other. This generalization of coloring has also been studied extensively [28, 2, 1]. As stated earlier, the problem of coloring paths is motivated by scheduling problems and it has been studied for many other variants, including call scheduling [12], nonrepetitive list colourings [16], *radio k-colorings* [8] etc.

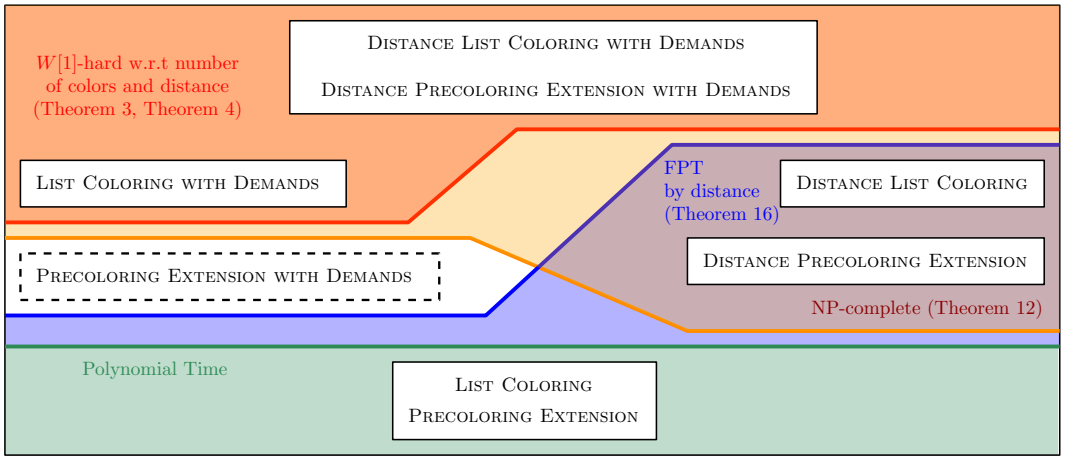


Figure 1 A pictorial representation of the results.

In this paper, we study the complexity of DPED when G is the union of disjoint paths. We prove that the decision version of the problem is NP-complete when the number of colors, the number of precolored vertices, and the minimum distance d between two vertices of the same color are part of the input. This hardness motivates the study of the problem in terms of parameterized complexity. We reduce the problem to the problem of List Coloring with Demands. We show that both problems are W[1]-hard when parameterized by distance and the number of colors. We prove that the hardness holds for the list coloring problem even if we drop the condition of distance coloring. However, the problem admits an FPT algorithm when we drop the demands and consider only the distance, implying that Distance Precoloring Extension on paths is also FPT by distance. Both problems remain NP-complete when the number of colors, the number of precolored vertices, and the distance are considered a part of the input. On the positive side, we prove that DPED is polynomial time solvable on a single path when the precolored vertices appear only at the ends. Further, we show that DPED on path instances is FPT when the number of colors, the distance parameter, and the number of precolored vertices are considered as parameters simultaneously. Then we present an approximation algorithm that runs in time $pd^{O(d)} + O(n \log c)$ and provides an additive approximate solution with an error of $O(d^2 \cdot p)$. Finally, we establish the NP-completeness of the problems of Distance List Coloring and Distance Precoloring extension on paths. The relationships between the problems and the results are illustrated in Figure 1. We pose the complexity of the Precoloring Extension with Demands on paths as an interesting open problem.

The paper is organized as follows. Section 2 presents a polynomial time solution for the DPED problem on a path having only its ends precolored. In Section 3 we establish the W[1]-hardness of DPED when the parameters are the number of colors and the distance. Section 4 contains results for special cases, when the problem parameters (number of precolored vertices, number of colors, distance) are in some sense limited: we present an approximation algorithm with small additive error and an exact FPT algorithm. Finally, Section 5 presents dynamic programming algorithm and NP-completeness results for the variant without the demands.

2 Polynomial algorithm for paths with precolored end-vertices

As our first result, we show that Distance Precoloring Extension with Demands can be solved in polynomial time whenever we allow precolored vertices only at the ends of the path. In this setting, we are coloring a path G with vertices $v_1, \ldots, v_n$ with a precoloring γ' that assigns colors to some initial segment $v_1, \ldots, v_s$ and some final segment $v_t, \ldots, v_n$. We say that such an instance of DPED is *end-precolored*. We describe a greedy algorithm that solves end-precolored instances of DPED in polynomial time, regardless of both the number of colors and the distance parameter.

The algorithm colors the vertices in their left-to-right order along the path, starting with the first uncolored vertex. For each vertex, it computes a set of feasible colors for the current vertex and chooses the one with the highest remaining demand. In case of a tie, the algorithm chooses the color that appears earlier in the precoloring of the final segment of the path. We remark that to compute the set of feasible colors for vertex v, it is necessary to exclude not only colors of the previous d vertices to the left of v but also the precolored endsegment in case v is at distance at most d from it. The technique is formally described in Algorithm 1.

Algorithm 1 Greedy algorithm for end-precolored DPED.

Input: End-precolored path instance (G, C, d, γ', η) of DPED, where G has vertices $v_1, \ldots, v_n$ and γ' precolors the segments $v_1, \ldots, v_s$ and $v_t, \ldots, v_n$.
Output: A d-distance coloring γ of G that extends γ' and satisfies the demands η, if it exists, and "`null`" otherwise.

```
for a ∈ C do                                   ▷ compute tie-breakers depending on the final segment
    pos(a) ← minimum j ≥ t such that γ'(v_j) = a, or ∞ if no such j exists.
γ ← γ'                                         ▷ initialize γ with the precoloring γ'
for i ← s + 1 to t − 1 do                      ▷ iterate over non-precolored vertices
    C_f ← C \ γ({v_max(i−d,1), ..., v_min(i+d,n)})   ▷ the set of feasible colors for v_i
    if C_f = ∅ or η(a) = 0 for every a ∈ C_f then
        return null                            ▷ no feasible color with positive demand
    a ← a color in C_f with the largest demand η(a) that, moreover, minimizes pos(a)
    γ(v_i) ← a                                 ▷ v_i receives the color a
    η(a) ← η(a) − 1                            ▷ demand of a is decreased
return γ
```

▶ **Theorem 1.** *Algorithm 1 solves DPED on end-precolored paths in time $O(n \log c)$ where n is the number of vertices and c is the number of colors.*

Proof. For contradiction, assume that Algorithm 1 fails, that is, there exists a feasible solution to the instance, and the algorithm does not find any. Let us denote the vertices of the path P as $v_1, v_2, \ldots, v_n$ from left to the right and let $R = \{v_t, \ldots, v_n\}$ be the right precolored segment of P. Let the algorithm produce a partial coloring γ and let γ' be a feasible solution where the index i satisfying $\gamma(v_j) = \gamma'(v_j)$ for all $j < i$ and $\gamma(v_i) \neq \gamma'(v_i)$ is the maximum possible. We are going to modify γ' such that we obtain another feasible solution that agrees with γ even on $\gamma(v_i)$.

Let $a = \gamma(i)$ and $b = \gamma'(i)$. By *alternating ba-sequence starting at* v_i we denote a maximal subsequence $S = (s_1, s_2, \ldots s_k)$ of the vertices $v_i, v_{i+1}, \ldots, v_n$ such that $s_1 = v_i$, $\gamma'(s_j) = b$ for odd j and $\gamma'(s_j) = a$ for even j, the distance of s_j and s_{j+1} is at most d for every

$j \in [k-1]$, and no vertex s_j is precolored. Observe that when the algorithm was assigning color to v_i, the remaining demand of the color a was at least the remaining demand of the color b, and that there is no color a in the distance less than d of the vertex v_i. Also note that for an alternating ba-sequence $s_1, \ldots, s_k$, there can be no vertex of P between any s_i and s_{i+1} colored by a or b, as this would break the distance property. Let us distinguish several cases.

First, consider the case when the last element of the sequence S is closer than d to the right precolored part R of P. In this case the sequence S must contain all occurrences of color a and b among the vertices from v_i to v_n in γ'. This implies that S is an even sequence, as the demand of a was at least the demand of b. If the vertices from R contain neither color a nor color b, we can freely swap the colors a and b in S. Suppose now the vertices from R contain color a (or b). If there is no occurrence of a or b in R that is within distance d of s_k, then the colors a and b can be swapped in γ'. If the first occurrence of a or b in R is within distance d of s_k, it must be b because s_k is colored a by γ', but this contradicts the fact that the algorithm has colored s_i with a instead of b.

Next, assume that the alternating ba-sequence S starting at v_i has even length and its last element is more than d vertices away from the precolored right part R of P. We can freely swap the colors a and b at the vertices of S as there is no conflict both to the left and right of S, thus obtaining a feasible solution γ'' with greater index i.

It remains to solve the case when S has odd length and its last element is more than d vertices far from the precolored right part R of P. The sequence S contains one more color b than a and as the algorithm has chosen the color a for v_i (which means the remaining demand of b was not greater than a), there must exists alternating ab-sequence S' starting at some vertex w of odd length, S' disjoint from S, which thus contains one more color a than b. We may assume that S' is not within distance d to the right precolored part R: if this is the case, we either obtain a contradiction if a occurs before b in R and a and b had the same demand, or we can recolor S and S' without problems. We may now swap the colors a and b both in S and S', which does not change the total number of colors used. This produces a feasible solution γ'' with greater index i.

We obtained the contradiction in all cases, which shows the correctness of the algorithms. The time complexity $O(n \log c)$ follows from using a smarter data structure for assigning the available colors: we may dynamically maintain a binary heap containing the set of feasible colors C_f with the key being a combination of the remaining demand with position (pos$(\cdot)$) and the value stored being the color label. ◀

3 Few colors and small distance

In this section, we focus on solving general instances of DPED when both the number of colors and the distance parameter are not too large. We stress that this additional assumption is very natural, e.g., for the original scheduling motivation.

First, we observe that we can solve DPED in this regime in polynomial time, provided that the number of colors and the distance parameter d are constant. The algorithm follows a standard dynamic programming approach for coloring and thus, we provide here only a brief sketch with all the details available in the full version.

▶ **Proposition 2.** *An instance (G, C, d, γ', η) of DPED on paths can be solved in time $c^{O(d)} \cdot n^{c+1}$ where n is the number of vertices and c is the number of colors.*

Proof sketch. Let γ_i be a d-distance coloring of an initial segment $\{v_1, \ldots, v_i\}$ of the path G. The signature of γ_i consists of the colors of the rightmost d vertices together with the frequencies of individual colors as used by γ_i. The algorithm sequentially computes for each

$i \in [n]$ all possible signatures of d-distance colorings of $\{v_1, \ldots, v_i\}$ that, moreover, extend the precoloring γ'. At the end, this suffices to check whether there is a d-distance coloring of $\{v_1, \ldots, v_n\}$ that extends γ' and the frequencies of colors used equal the demands. ◀

It is only natural to ask whether we can achieve a polynomial runtime in this regime where the degree of the polynomial is independent of both c and d. We answer this question negatively, under standard theoretical assumptions, by showing that DPED is W[1]-hard with respect to both of these parameters.

▶ **Theorem 3.** *DPED is* W[1]*-hard on paths with respect to the number of colors and the distance d.*

We first show W[1]-hardness of a related coloring problem, namely List Coloring with Demands. We define the problem formally as follows.

List Coloring with Demands (LCD)
Input: A graph $G = (V, E)$ on v vertices, a set of colors C, a function $L\colon V \to 2^C$ that assigns a list of colors $L(v)$ to each vertex, and a demand function $\eta\colon C \to [n]$.
Question: Is there an L-list coloring $\gamma\colon V \to C$ that satisfies the demands η?

An instance (G, C, η, L) of LCD is a *path instance* if the graph G is a disjoint union of paths. Moreover, a path instance of LCD is *non-alternating* if there does not exist a color $c \in C$ and three consecutive vertices v_{i-1}, v_i, v_{i+1} on some path such that $c \notin L(v_i)$ and simultaneously $c \in L(v_{i-1}) \cap L(v_{i+1})$.

The LCD problem has already been shown to be W[1]-hard on paths with respect to the number of colors by Gomes, Guedes, and dos Santos [15] albeit under a different name of Number List Coloring. However, the produced instances of LCD therein are very far from having the non-alternating property that is crucial in proving Theorem 3. We devise a novel reduction from a multidimensional version of the subset problem that, moreover, produces non-alternating path instances of LCD. Due to the space constraints, we omit the proof which can be found in the full version.

▶ **Theorem 4.** *LCD is* W[1]*-hard with respect to the number of colors even when restricted to non-alternating path instances.*

Theorem 3 is obtained through a polynomial-time reduction from LCD on non-alternating path instances. Crucially, both the distance parameter and the number of colors in the produced DPED instance are linear in the number of colors in the original LCD instance.

Proof of Theorem 3. For technical reasons, we first modify the LCD instance to guarantee that it is a single path, and both of its endpoints have a list identical to its only neighbor v. That is achieved by introducing two extra colors a, b and concatenating all paths in arbitrary order with two extra vertices in between consecutive pairs and two extra vertices at the very beginning and end where we add $\{a, b\}$ to the list of all vertices and set $L(v) = \{a, b\}$ for each new vertex. Finally, we set $\eta(a) = \eta(b) = p + 1$ where p is the total number of paths in G. It is easy to see that the modified instance is equivalent. Furthermore, the modified instance remains non-alternating because there are two extra new vertices between any pair of original paths, and the colors a, b are included in the list of every vertex.

From now on, we assume that G is a single path with vertices $v_1, \ldots, v_n$ and edges $e_1, \ldots, e_{n-1}$ such that $e_i = \{v_i, v_{i+1}\}$ for each $i \in [n-1]$. Moreover, we have $L(v_1) = L(v_2)$ and $L(v_{n-1}) = L(v_n)$.

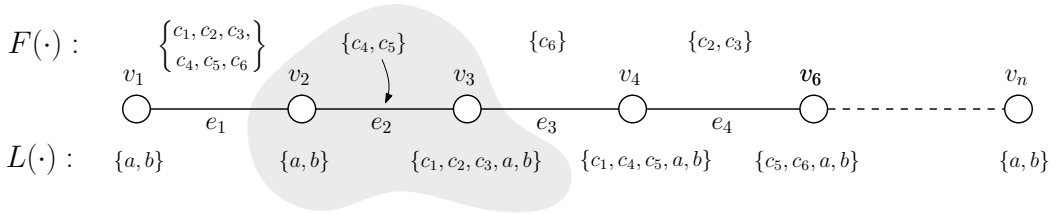


Figure 2 Representation of LCD with forbidden colors when $C = \{c_1, c_2, \ldots, c_6, a, b\}$.

Now, we show that the lists of such a non-alternating LCD path instance can be alternatively represented by lists of forbidden colors for each edge. It is crucial that the instance of LCD is non-alternating, as otherwise, this may not be possible. Moreover, we will also guarantee that any fixed color can be contained in the forbidden sets of at most two consecutive edges.

▷ Claim 5. We can compute in polynomial time an assignment of color sets to edges $F\colon E(G) \to 2^C$ such that for every vertex v, we have $L(v) = C \setminus (F(e) \cup F(e'))$ where e, e' are the two edges incident to v, or $L(v) = C \setminus F(e)$ if v is incident to a single edge e. Moreover, there are no three consecutive edges e_{i-1}, e_i and e_{i+1} such that the intersection $F(e_{i-1}) \cap F(e_i) \cap F(e_{i+1})$ is non-empty.

Proof. We define the assignment inductively along the path. We set $F(e_1) = C \setminus L(v_1)$ and $F(e_{n-1}) = C \setminus L(v_n)$. Recall that we first modified the instance to guarantee that each endpoint of the path has an identical list to its neighbor, i.e., $L(v_1) = L(v_2)$ and $L(v_{n-1}) = L(v_n)$. For $i \in \{2, \ldots, n-2\}$, we let $F(e_i)$ contain a color $c \in C$ if and only if $c \notin L(v_i) \cup L(v_{i+1})$ and moreover, either (i) $c \notin F(e_{i-1})$, or (ii) $c \in L(v_{i+2})$. See Figure 2. Clearly, F can be computed in polynomial time.

Let us verify the properties of F. For the endpoints v_1 and v_n, we have $L(v_1) = C \setminus F(e_1)$ and $L(v_n) = C \setminus F(e_{n-1})$ by definition. Now, fix arbitrary $i \in \{2, \ldots, n-1\}$ and a color $c \in C$. If we have $c \in L(v_i)$, then neither $F(e_{i-1})$ nor $F(e_i)$ contain c by definition and $c \in C \setminus (F(e_{i-1}) \cup F(e_i))$. On the other hand, if $c \notin L(v_i)$, it follows from the non-alternating property that either $c \notin L(v_{i-1})$ or $c \notin L(v_{i+1})$. We consider the three possible cases. First, assume that c appears in none of the lists $L(v_{i-1})$, $L(v_i)$, and $L(v_{i+1})$. Then either $c \in F(e_{i-1})$ or we have $c \notin F(e_{i-1})$ which triggers the condition (i) in the definition of $F(e_i)$ and either way, we get that $c \in F(e_{i-1}) \cup F(e_i)$. Second, assume that c appears in the list $L(v_{i+1})$ but not in $L(v_{i-1})$. In this case, we have $c \in F(e_{i-1})$ by condition (ii) in the definition of $F(e_{i-1})$ and again, we see that $c \in F(e_{i-1}) \cup F(e_i)$. Finally, assume that c appears in the list $L(v_{i-1})$ but not in $L(v_{i+1})$. In this case, we have $c \notin F(e_{i-1})$, which implies $c \in F(e_i)$ by condition (i) in the definition. In all cases, we obtained $c \in F(e_{i-1}) \cup F(e_i)$.

Assume for a contradiction that there are three consecutive edges e_{i-1}, e_i and e_{i+1} such that the intersection $F(e_{i-1}) \cap F(e_i) \cap F(e_{i+1})$ is non-empty and let c be an arbitrary color in the intersection. However, observe that in the definition of $F(e_i)$ neither condition (i) nor (ii) holds for c and thus, we must have $c \notin F(e_i)$. ◁

With this assignment F, we are finally ready to define the path instance $(G', C', d, \gamma', \eta')$ of DPED. Let t denote the size of the color set C and let $C = \{c_1, \ldots, c_t\}$ be an arbitrary enumeration of the colors. We set $d = 2t + 1$. We take the graph G' to be a path of length $nd + 1$ on vertices $v'_1, \ldots, v'_{nd+1}$ in this order along the path. To simplify the exposition, we partition the vertices into two distinct sets. Every vertex $v'_{(k-1)d+1}$ for some $k \in [n]$ is a *main vertex* and we denote it alternatively by x_k. All the remaining vertices are *auxiliary*.

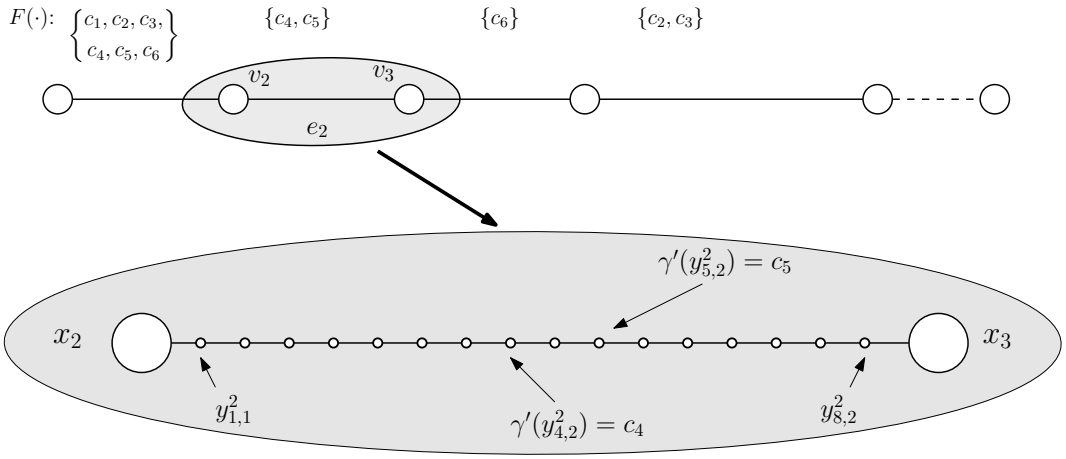


Figure 3 Construction of G' from G. Extending from Figure 2, $\gamma'(y^2_{j,\alpha}) = \gamma^\star(y^2_{j,\alpha})$, except for $y^2_{4,2}$ and $y^2_{5,2}$.

For $i \in [n-1]$, $j \in [t]$ and $\alpha \in [2]$, the vertex $v'_{(i-1)d+2j+\alpha-1}$ is an auxiliary vertex *associated to color* c_j and we alternatively denote it by $y^i_{j,\alpha}$. Observe that every two consecutive main vertices x_i and x_{i+1} are separated by exactly $2t$ auxiliary vertices that are grouped into pairs associated to colors $c_1, \ldots, c_t$ in this order. More specifically, the segment between x_i and x_{i+1} contains the auxiliary vertices $y^i_{j,\alpha}$ for all $j \in [t]$ and $\alpha \in [2]$, ordered lexicographically by the pairs (j, α). See Figure 3.

Our goal is to define a pre-coloring of all the auxiliary vertices such that there is a one-to-one correspondence between feasible colorings of the main vertices and feasible colorings of the original LCD instance. As a final ingredient, we need to be able to assign extra colors to certain vertices without affecting the rest of the instance. To that end, we set $C' = C \cup C^\star$ where $C^\star = \{c^\star_0, \ldots, c^\star_d\}$ is a set of *auxiliary colors* disjoint from C. Moreover, we define an auxiliary coloring $\gamma^\star \colon V(G') \to C^\star$ where $\gamma^\star(v'_i) = c^\star_{i \bmod (d+1)}$ for each $i \in [nd+1]$. Informally, the coloring $\gamma^\star$ simply cyclically iterates through the sequence of auxiliary colors $c^\star_0, \ldots, c^\star_d$ along the path, starting with $c^\star_1$. Clearly, $\gamma^\star$ is a proper d-distance coloring of G'.

We define the pre-coloring γ' of every auxiliary vertex. For $i \in [n-1]$ and $j \in [t]$, we set

$$\gamma'(y^i_{j,1}) = \begin{cases} c_j & \text{if } c_j \in F(e_i) \text{ and } c_j \notin F(e_{i-1}). \\ \gamma^\star(y^i_{j,1}) & \text{otherwise.} \end{cases}$$

$$\gamma'(y^i_{j,2}) = \begin{cases} c_j & \text{if } c_j \in F(e_i) \cap F(e_{i-1}). \\ \gamma^\star(y^i_{j,2}) & \text{otherwise.} \end{cases}$$

where we additionally define $F(e_0) = \emptyset$ for simplicity.

First, observe that the pre-coloring γ' exactly encodes the assignment of forbidden colors to edges.

▶ **Observation 6.** *A color $c_j \in C$ appears in the pre-coloring γ' on the segment between main vertices x_i and x_{i+1} if and only if $c_j \in F(e_i)$.*

Moreover, we show that γ' distributes the occurrences of a fixed color at a sufficient distance from each other.

▷ Claim 7. The pre-coloring γ' does not contain any monochromatic pair of vertices at a distance at most d.

Proof. Assume for a contradiction that γ' does contain such a pair of monochromatic vertices. We have already observed that $\gamma^\star$ is a proper d-coloring on a disjoint set of auxiliary colors and thus, this pair must use some color $c_j \in C$. We cannot have $\gamma'(y^i_{j,1}) = \gamma'(y^i_{j,2}) = c_j$ for any fixed i, j since the respective conditions in the definition of γ' are mutually exclusive.

For any different $i, i' \in [n-1]$ with $i < i'$ and arbitrary $\alpha, \alpha' \in [2]$, the distance between $y^i_{j,\alpha}$ and $y^{i'}_{j,\alpha'}$ is at most d if and only if $i' - i = 1$ and $\alpha' \leq \alpha$. We cannot have $\gamma'(y^i_{j,1}) = \gamma'(y^{i+1}_{j,1}) = c_j$ by definition of $\gamma'(y^{i+1}_{j,1})$. So the only option left is that $\gamma'(y^i_{j,2}) = \gamma'(y^{i+1}_{j,\alpha}) = c_j$ for either $\alpha \in [2]$. But in that case, we have $c_j \in F(e_{i-1}) \cap F(e_i)$ since $\gamma'(y^i_{j,2}) = c_j$ and simultaneously, $c_j \in F(e_{i+1})$ since $\gamma'(y^{i+1}_{j,\alpha}) = c_j$. Thus, we reached a contradiction since F does not contain the same color in the sets $F(e_{i-1}), F(e_i), F(e_{i+1})$ of three consecutive edges. ◁

To finish the construction, it remains to define the demand function η'. We set $\eta'(c_j) = \eta(c_j)$ for every original color $c_j \in C$ and we set $\eta'(c^\star_j) = 0$ for every auxiliary color $c^\star_j \in C^\star$.

Correctness ("only if"). Suppose that (G, C, η, L) is a yes-instance of LCD and let $\rho\colon V(G) \to C$ be an L-list coloring that meets the demands. We define a coloring $\gamma\colon V(G') \to C'$ by simply setting $\gamma(x_i) = \rho(v_i)$ for every main vertex x_i and $\gamma(y^i_{j,\alpha}) = \gamma'(y^i_{j,\alpha})$ for every auxiliary vertex $y^i_{j,\alpha}$. The coloring γ is an extension of γ' by definition, and it meets the demands because the demand functions η and η' are equal when restricted to the color set C. Thus, it remains to show that γ is a proper d-distance coloring.

Assume for a contradiction that γ contains a pair of monochromatic vertices u and w at a distance at most d. At least one of these vertices has to be a main vertex since the colors of all auxiliary vertices are fixed by γ' and there is no such monochromatic pair of pre-colored vertices by Claim 7. First, assume that both u and w are main vertices. Two main vertices are in distance at most d if and only if they are consecutive, i.e., we have without loss of generality $u = x_i$ and $w = x_{i+1}$ for some $i \in [n-1]$. But then they cannot share the same color since $\gamma(x_i) = \rho(v_i)$, $\gamma(x_{i+1}) = \rho(v_{i+1})$ and ρ is a proper list-coloring of G. Otherwise, suppose without loss of generality that u is a main vertex x_i and w is an auxiliary vertex. The coloring γ can use only the colors from C on the main vertices and thus, we have $\gamma(x_i) = \gamma(w) = c_j$ for some $c_j \in C$. Moreover, the vertex w lies either on the segment between x_{i-1} and x_i or on the segment between x_i and x_{i+1} as those are the only auxiliary vertices at a distance at most d from x_i. Observation 6 then implies that we have either $c_j \in F(e_{i-1})$ or $c_j \in F(e_i)$. But we know that $L(x_i) = C \setminus (F(e_{i-1}) \cap F(e_i))$ by Claim 5. In particular, the color c_j does not appear in the list $L(v_i)$, which is a contradiction with ρ being a proper list-coloring since $\rho(v_i) = \gamma(x_i) = c_j$.

Correctness ("if"). Now suppose that $(G', C', d, \gamma', \eta')$ is a yes-instance of DPED and let $\gamma\colon V(G') \to C'$ be a witnessing d-distance coloring, i.e., γ extends the pre-coloring γ' and meets the demands given by η'. We define a coloring $\rho\colon V(G) \to C$ by simply restricting γ to the main vertices, that is we set $\rho(v_i) = \gamma(x_i)$. First, observe that the range of ρ is indeed only the color set C since the demand $\eta'(c^\star_j)$ of any auxiliary color is zero and thus, γ uses only the colors from C on the main vertices. Moreover, we have $\rho(v_i) \neq \rho(v_{i+1})$ for every $i \in [n-1]$ because the main vertices x_i and x_{i+1} are at distance d in G'. The demands are also satisfied by ρ because the demand functions η and η' are equal when restricted to the color set C.

It remains to check that ρ is an L-list coloring, i.e., we have $\rho(v_i) \in L(v_i)$ for all $i \in [n]$. Assume for a contradiction that this does not hold for a vertex v_i with $\rho(v_i) = \gamma(x_i) = c_j$. By Claim 5, the color c_j belongs to at least one of the sets $F(e_{i-1})$ and $F(e_i)$. Observation 6

implies that the color c_j is used by γ on some auxiliary vertex from the segment between x_{i-1} and x_{i+1}. But all these vertices are at a distance at most d from x_i and we reach a contradiction with γ being a proper d-distance coloring. ◀

4 Few precolored vertices

In this section, we consider the regime when we are given only a few precolored vertices. We present two effective algorithms for DPED in this regime. We first show that an approximation FPT algorithm with small additive error is possible even in the case of many colors. Afterwards, we present an exact FPT algorithm when the number of colors is small as well.

4.1 Approximation

Next, we show there is an approximation algorithm for DPED with additive error, which is bounded by a polynomial function of the distance parameter and the number of colors. By having additive error α we mean that for the set of colors C and demands $\eta\colon C \to [n]$ the algorithm produces coloring $\gamma\colon V \to C$ such that

$$\sum_{c \in C} \left|\eta(c) - |\gamma^{-1}(c) \setminus A|\right| \leq \alpha.$$

▶ **Theorem 8.** *Let us have an instance of DPED on path P of length n with p precolored vertices, and let $c = |C| \geq d + 2$. There is an approximation algorithm that runs in time $pd^{O(d)} + O(n \log c)$ and outputs a coloring with an additive error at most $O(pd^2)$.*

Proof. The approximation algorithm is as follows. We ignore all precolored vertices and we run the greedy Algorithm 1 to color the path P. Let the precolored vertices be denoted $w_1, \ldots, w_p$, respectively, along the path P, which consists of vertices $v_1, \ldots, v_n$ from left to right. Let $b = 2(d+1)^2$. Then we remove the colors in the neighborhood of diameter b around each precolored vertex w_i (that is, for the vertices v_j whose distance to some w_i is at most b). The idea is now to reassign the colors to these discolored vertices while introducing only a small additive error.

First we handle all pairs of vertices (w_i, w_{i+1}) such that $\mathrm{dist}(w_i, w_{i+1}) \leq 2b$ as follows. If $|C| \geq 2d + 1$, it is easy to find the coloring of the gap between w_i and w_{i+1} in a greedy manner. If $|C| < 2d + 1$, we produce d-distance coloring of each segment of consecutive gaps of size at most $2b$ by dynamic programming in time $pd^{O(d)}$. We may thus further assume that all remaining uncolored gaps between pairs of vertices (w_i, w_{i+1}) have $\mathrm{dist}(w_i, w_{i+1}) > 2b$. For a precolored vertex w_i, which equals to some v_j, let us denote by *left neighborhood of* w_i those vertices v_k, where $\mathrm{dist}(v_k, v_j) \leq b$ and $k < j$. Similarly, we define the right neighborhood as vertices v_k where $\mathrm{dist}(v_k, v_j) \leq b$ and $k > j$. We now describe the procedure that colors the diameter b neighborhood of each w_i, first the left neighborhood and then the right neighborhood (provided that the respective neighborhood is still uncolored). Observe that if the right neighborhood of w_i and left neighborhood of w_{i+1} are both uncolored then they are disjoint as $\mathrm{dist}(w_i, w_{i+1}) > 2b$. We split the b vertices of the left neighborhood into $2(d+1)$ blocks $B_1, \ldots, B_{2(d+1)}$ of size $d+1$. Let us denote the $d+1$ vertices immediately preceding block B_1 as B_0 and let the $d+1$ vertices after $B_{2(d+1)}$ be denoted as B_{2d+3}. Here, the vertex w_i is the first vertex of B_{2d+3} and we choose the coloring of B_{2d+3} to consist of the precoloring of w_i followed by arbitrarily chosen valid coloring of the rest of B_{2d+3}. Denote by B_i^j the color of the vertex u_j of the block $B_i = u_1, \ldots, u_{d+1}$.

We proceed by partial modification of the coloring of each block B_i into the block B_{i+1}, for $i = 0, \ldots, 2d+1$. Let the coloring of B_0 be, without loss of generality, $0, 1, \ldots, d$. For the block B_1, we copy the coloring from B_0, where we remove the occurrence of color B^1_{2d+3}, shift the colors to the right, and introduce a new first color to be $d+1$ (that is, an extra color). For the block B_2, we copy the coloring from B_1, where we replace the first color $d+1$ into B^1_{2d+3}. Note that by these two steps, we did not create any color conflict for vertices closer than d. For $i = 2, \ldots, d+1$ we now continue in a similar manner: let the coloring of the block B_{2i-1} be copied from the block B_{2i-2}, where we remove the occurrence of the color B^i_{2d+3}, shift the colors to the right and set B^i_{2i-1} to d. The coloring of B_{2i} is copied from B_{2i-1}, but we set B^i_{2i} to B^i_{2d+3}. Note that the number of blocks is enough to correctly replace all colors such that at the end the coloring of the block B_{2d+3} is reached and no coloring conflict is created.

After the coloring of the left neighborhood of w_i, we proceed similarly with the right neighborhood (with the difference that the block B_{d+2} may now inherit the coloring from the first (greedy) phase. The total running time is clearly $pd^{O(d)} + O(n \log c)$. The additive error introduced by this coloring routine is at most the size of all neighborhoods of the precolored vertices $w_1, \ldots, w_p$, which is $O(pd^2)$. ◀

4.2 Exact algorithm for few colors

We have already seen that DPED is W[1]-hard when parameterized by both the number of colors and the distance, which rules out the existence of an FPT algorithm parameterized by these parameters alone under standard assumptions. We have also seen that an FPT approximation algorithm with additive error is possible if there are few precolored vertices and the distance parameter is small, even for an unbounded number of colors. Here, we build on these results and show that an exact FPT algorithm is possible when the number of colors, the distance parameter, and the number of precolored vertices are all small. In some ways, it can also be seen as an extension of the greedy algorithm from Theorem 1 in the case of few colors and small distance. Our approach exploits an intriguing connection between d-distance colorings of paths and regular languages.

A *non-deterministic finite automaton (NFA)* A is a tuple $(\Sigma, Q, \delta, q_0, F)$, where Σ is a finite alphabet, Q is a finite set of states, $q_0 \in Q$ is an initial state, $F \subseteq Q$ is the set of final states, and $\delta \subseteq Q \times \Sigma \times Q$ is a transition relation. A word $w = w_1 \cdots w_n$ over the alphabet Σ is *accepted by* A if there exists a sequence of states $p_0, \ldots, p_n$ such that p_0 is the initial state q_0, p_n is some final state in F and we have $(p_{i-1}, w_i, p_i) \in \delta$ for every $i \in [n]$. We denote by $L(A)$ the language of all words accepted by A.

For a word $w \in \Sigma^*$ and a letter $\alpha \in \Sigma$, we let $|w|_\alpha$ denote the number of occurrences of α in w. We assume that $\Sigma = \{\alpha_1, \ldots, \alpha_k\}$ and we define the *Parikh image* $\mathcal{P}(w)$ of a word $w \in \Sigma^*$ to be the tuple $(|w|_{\alpha_1}, \ldots, |w|_{\alpha_k}) \in \mathbb{N}^k$. In other words, the Parikh image of a word w captures exactly the number of occurrences of each letter from Σ in w. Moreover, for a set of words $L \subseteq \Sigma^*$, we let $\mathcal{P}(L)$ denote the set of all Parikh images of words in L.

It is known that given a small NFA A over a small alphabet Σ together with a tuple $\mathbf{b}$, it is possible to efficiently decide whether there exists a word in $w \in L(A)$ such that $\mathcal{P}(w) = \mathbf{b}$.

▶ **Theorem 9** ([21, 31]). *Given an NFA $\mathcal{A}$ with n states over the alphabet Σ of size k, and a tuple $\mathbf{b} \in \mathbb{N}^k$, checking whether $\mathbf{b} \in \mathcal{P}(L(\mathcal{A}))$ can be done in time $2^{O(k^2 \log(kn) + \log\log b)}$ where $b = \|\mathbf{b}\|_\infty$.*

The main theorem of this section follows by reducing DPED to membership for Parikh images of a regular language given by a small NFA. In order to make the argument easier to follow, we use an intermediate problem that asks whether there exists a word in the

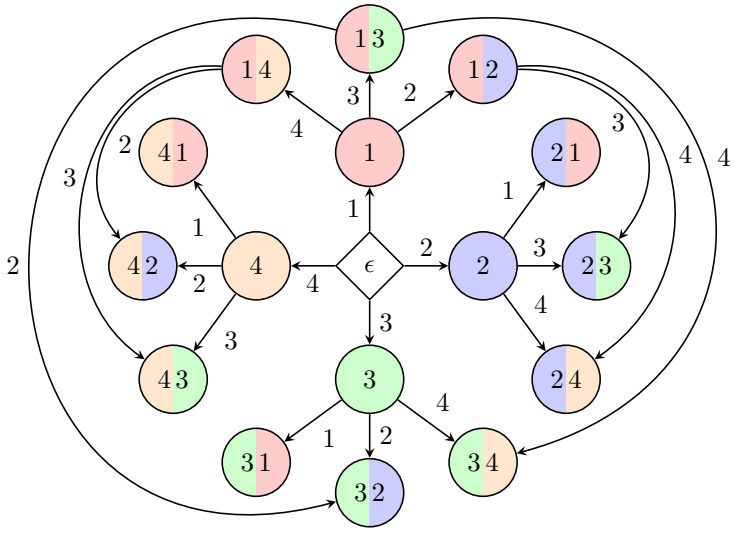


Figure 4 The NFA $A_{[4],2}$ that accepts exactly the 2-distance colorings of paths using colors $\{1, \ldots, 4\}$. The initial state ϵ is in the center, all its states are final and for improved readability, we omit all transitions from states with two colors except for 12, 13 and 14.

regular language with a given Parikh image that, moreover, has predetermined letters in some positions. Let P be a relation $P \subseteq \mathbb{N} \times \Sigma$ where Σ is a finite alphabet. We say that a word w is *P-constrained* if we have $w_i = \alpha$ for every $(i, \alpha) \in P$.

Constrained Membership for Parikh Languages of NFAs (CMPL)
Input: An NFA A over alphabet Σ of size k, a tuple $\mathbf{b} \in \mathbb{N}^k$ and a relation $P \subseteq \mathbb{N} \times \Sigma$.
Question: Is there a P-constrained word $w \in L(A)$ such that $\mathcal{P}(w) = \mathbf{b}$?

▶ **Theorem 10.** *CMPL can be solved in time $2^{O((k+p)^2 \log((k+p)n)+\log\log b)}$ where n is the number states of the NFA A, $b = \|\mathbf{b}\|_\infty$ and $p = |P|$.*

Due to the space constraints, we omit the proof of Theorem 10.

▶ **Theorem 11.** *DPED can be solved on path instances in time $O(n) + 2^{O(d\cdot(c+q)^2 \log(c+q))} \cdot (\log n)^{O(1)}$ where c is the number of colors, d is the distance parameter and q is the number of precolored vertices.*

Proof. Let (G, C, d, γ', η) be an instance of DPED. As we have observed before, we can assume that G is in fact a single path on vertices $v_1, \ldots, v_n$ in this order along the path.

We start by showing that proper d-distance colorings of paths form a regular language over C. But first, we need some notation. For an alphabet Σ and a positive integer t, we let $\Sigma^{\preceq t}$ denote the set of all words of length at most t over Σ with no repeated letters, including the empty word ε. Moreover for word $w \in \Sigma^{\preceq t}$ and letter $\alpha \notin w$, we let $\psi_t(w, \alpha)$ be an operation which appends the letter α at the end of w and then possibly deletes the first letter if w already has length t.

Now we can define the NFA $A_{\Sigma,t} = (\Sigma, \Sigma^{\preceq t}, \delta, \varepsilon, \Sigma^{\preceq t})$ where $(w, \alpha, w') \in \delta$ if and only if $\psi_t(w, \alpha) = w'$. The language $L(A_{\Sigma,t})$ clearly contains exactly those words over Σ that do not contain two occurrences of the same letter at a distance less than $t + 1$ from each other. See Figure 4. Therefore, the NFA $A_{C,d}$ accepts exactly valid d-distance colorings of paths, and we can solve DPED with no precolored vertices by invoking Theorem 9 for the NFA $A_{C,d}$ where the tuple $\mathbf{b}$ encodes the demand function η.

Moreover, we can encode the precolored vertices as additional constraints. In particular, we define a set of constraints P where $(i, \alpha) \in P$ if and only if the vertex v_i is precolored and $\gamma'(v_i) = \alpha$. Additionally, we set $\mathbf{b} \in \mathbb{N}^c$ where $b_j = \eta(j) + |\gamma'^{-1}(j)|$ for each $j \in [c]$. Observe that the mapping $w \mapsto \gamma_w$ where $\gamma_w(v_i) = w_i$ is a bijection between P-constrained words with $\mathcal{P}(w) = \mathbf{b}$ and proper distance d-colorings of G that extend γ' and satisfy the demands η. In other words, (G, C, d, γ', η) is yes-instance of DPED if and only if $(A_{C,d}, \mathbf{b}, P)$ is a yes-instance of CMPL.

To finish the proof, it remains to invoke the algorithm of Theorem 10 on the produced NFA A, the tuple $\mathbf{b}$, and the set of constraints P. The desired runtime follows since the automaton $A_{C,d}$ has $O(c^d)$ states and the alphabet is exactly the set of colors C of size c. ◀

5 Coloring without demands

In this section, we focus on variants of the two problems (Precoloring Extension and List Coloring) on paths where we no longer specify demands for each color, but we still require vertices at distance at most d to receive different colors. We refer to these problems as Distance Precoloring Extension (DPE) and Distance List Coloring (DLC). Due to the space constraints, we defer most proofs from this section to the full version.

We show that DPE is NP-complete on paths by reduction from Precoloring Extension on unit interval graphs [24]. We remark that unit interval graphs are a suitable source of hardness since it is known that unit interval graph are precisely the induced subgraphs of path powers [23]. As a consequence, we obtain the NP-completeness of DPED on paths.

▶ **Theorem 12.** *DPE is* NP*-complete even when restricted to paths.*

▶ **Corollary 13.** *DPED is* NP*-complete even when restricted to paths.*

Proof of Corollary 13. Let (G, C, d, γ') be an instance of DPE where G is a path with n vertices. We construct an instance $(G', C, d, \gamma', \eta)$ where G' is simply obtained by adding $(|C| - 1) \cdot n$ isolated vertices (paths of zero length) to G and setting $\eta(a) = n$ for every $a \in C$. Clearly, every d-distance coloring of G' that extends γ' induced the desired d-distance coloring of G, regardless of the demands. On the other hand, we can extend any d-distance coloring γ of G to G' by coloring exactly $n - |\gamma^{-1}(a)|$ of the isolated vertices with each color $a \in C$ to satisfy all demands.

Finally, we construct an equivalent instance of DPED on a single path. It suffices to concatenate all the disjoint paths in G' to a single path while adding d auxiliary vertices between each consecutive pair of original paths. These auxiliary vertices are then greedily precolored with $2d + 1$ new auxiliary colors disjoint from C that have zero demands. In this way, the d-distance colorings of individual paths from G' are independent from each other, and the equivalence follows. ◀

In contrast, we show that the problem Distance Precoloring Extension without demands becomes FPT by the distance parameter d even for an unbounded number of colors. Recall that Distance Precoloring Extension with Demands is W[1]-hard with respect to d by Theorem 3. In fact, we show the existence of an FPT algorithm for the more general problem of Distance List Coloring where the task is to find a list coloring that assigns different colors to every pair of vertices at distance at most d.

▶ **Theorem 14.** *Distance List Coloring can be solved on paths in time $d^{O(d)} \cdot n$ where n is the number of vertices.*

References

1 Geir Agnarsson, Raymond Greenlaw, and Magnús M Halldórsson. On powers of chordal graphs and their colorings. *Congressus Numerantium*, pages 41–66, 2000.

2 Geir Agnarsson and Magnús M. Halldórsson. Coloring powers of planar graphs. *SIAM J. Discret. Math.*, 16(4):651–662, 2003. doi:10.1137/S0895480100367950.

3 Leonid Barenboim, Michael Elkin, and Fabian Kuhn. Distributed ($\Delta + 1$)-coloring in linear (in Δ) time. *SIAM J. Comput.*, 43(1):72–95, 2014. doi:10.1137/12088848X.

4 Sayan Bhattacharya, Deeparnab Chakrabarty, Monika Henzinger, and Danupon Nanongkai. Dynamic algorithms for graph coloring. 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 1–20. SIAM, 2018. doi:10.1137/1.9781611975031.1.

5 Miklós Biró, Mihály Hujter, and Zsolt Tuza. Precoloring extension. I. Interval graphs. *Discret. Math.*, 100(1-3):267–279, 1992. doi:10.1016/0012-365X(92)90646-W.

6 Hans L. Bodlaender, Carla Groenland, and Hugo Jacob. List colouring trees in logarithmic space. In Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman, editors, *30th Annual European Symposium on Algorithms, ESA 2022, September 5-9, 2022, Berlin/Potsdam, Germany*, volume 244 of *LIPIcs*, pages 24:1–24:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ESA.2022.24.

7 Hajo Broersma, Petr A. Golovach, Daniël Paulusma, and Jian Song. On coloring graphs without induced forests. In Otfried Cheong, Kyung-Yong Chwa, and Kunsoo Park, editors, *Algorithms and Computation – 21st International Symposium, ISAAC 2010, Jeju Island, Korea, December 15-17, 2010, Proceedings, Part II*, volume 6507 of *Lecture Notes in Computer Science*, pages 156–167. Springer, 2010. doi:10.1007/978-3-642-17514-5_14.

8 Gary Chartrand, Ladislav Nebesky, and Ping Zhang. Radio k-colorings of paths. *Discuss. Math. Graph Theory*, 24(1):5–21, 2004. doi:10.7151/DMGT.1209.

9 Guilherme de C. M. Gomes, Carlos V. G. C. Lima, and Vinícius Fernandes dos Santos. Parameterized complexity of equitable coloring. *Discret. Math. Theor. Comput. Sci.*, 21(1), 2019. doi:10.23638/DMTCS-21-1-8.

10 Marc Demange, Tínaz Ekim, Bernard Ries, and Cerasela Tanasescu. On some applications of the selective graph coloring problem. *Eur. J. Oper. Res.*, 240(2):307–314, 2015. doi:10.1016/J.EJOR.2014.05.011.

11 Paul Erdős, Arthur L Rubin, and Herbert Taylor. Choosability in graphs. *Congr. Numer*, 26(4):125–157, 1979.

12 Thomas Erlebach and Klaus Jansen. The complexity of path coloring and call scheduling. *Theor. Comput. Sci.*, 255(1-2):33–50, 2001. doi:10.1016/S0304-3975(99)00152-8.

13 Michael R. Fellows, Fedor V. Fomin, Daniel Lokshtanov, Frances A. Rosamond, Saket Saurabh, Stefan Szeider, and Carsten Thomassen. On the complexity of some colorful problems parameterized by treewidth. *Inf. Comput.*, 209(2):143–153, 2011. doi:10.1016/J.IC.2010.11.026.

14 Robert Freimer. Generalized map coloring for use in geographical information systems. In Ki-Joune Li, Kia Makki, Niki Pissinou, and Siva Ravada, editors, *ACM-GIS 2000, Proceedings of the Eighth ACM Symposium on Advances in Geographic Information Systems, November 10-11, 2000, Washington D.C., USA*, pages 167–173. ACM, 2000. doi:10.1145/355274.355299.

15 Guilherme C. M. Gomes, Matheus R. Guedes, and Vinícius Fernandes dos Santos. Structural parameterizations for equitable coloring: Complexity, FPT algorithms, and kernelization. *Algorithmica*, 85(7):1912–1947, 2023. doi:10.1007/S00453-022-01085-W.

16 Jaroslaw Grytczuk, Jakub Przybylo, and Xuding Zhu. Nonrepetitive list colourings of paths. *Random Struct. Algorithms*, 38(1-2):162–173, 2011. doi:10.1002/RSA.20347.

17 Gregory Z. Gutin, Diptapriyo Majumdar, Sebastian Ordyniak, and Magnus Wahlström. Parameterized pre-coloring extension and list coloring problems. *SIAM J. Discret. Math.*, 35(1):575–596, 2021. doi:10.1137/20M1323369.

18 Lars Jaffke and Bart M. P. Jansen. Fine-grained parameterized complexity analysis of graph coloring problems. *Discret. Appl. Math.*, 327:33–46, 2023. doi:10.1016/j.dam.2022.11.011.

19 Tommy R. Jensen and Bjarne Toft. *Graph Coloring Problems*. Wiley Series in Discrete Mathematics and Optimization. Wiley, 2011.

20 Susan Khor. Application of graph colouring to biological networks. *IET Systems Biology*, 4:185–192, 2010. doi:10.1049/iet-syb.2009.0038.

21 Eryk Kopczynski and Anthony Widjaja To. Parikh images of grammars: Complexity and applications. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom*, pages 80–89. IEEE Computer Society, 2010. doi:10.1109/LICS.2010.21.

22 Jan Kratochvíl, Zsolt Tuza, and Margit Voigt. New trends in the theory of graph colorings: Choosability and list coloring. In Ronald L. Graham, Jan Kratochvíl, Jaroslav Nešetřil, and Fred S. Roberts, editors, *Contemporary Trends in Discrete Mathematics: From DIMACS and DIMATIA to the Future, Proceedings of a DIMACS Workshop, Stirín Castle, Czech Republic, May 19-25, 1997*, volume 49 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 183–197. DIMACS/AMS, 1997. doi:10.1090/dimacs/049/13.

23 Min Chih Lin, Dieter Rautenbach, Francisco J. Soulignac, and Jayme Luiz Szwarcfiter. Powers of cycles, powers of paths, and distance graphs. *Discret. Appl. Math.*, 159(7):621–627, 2011. doi:10.1016/J.DAM.2010.03.012.

24 Dániel Marx. Precoloring extension on unit interval graphs. *Discret. Appl. Math.*, 154(6):995–1002, 2006. doi:10.1016/j.dam.2005.10.008.

25 Walter Meyer. Equitable coloring. *The American mathematical monthly*, 80(8):920–922, 1973.

26 Hiroki Osawa, Akira Suzuki, Takehiro Ito, and Xiao Zhou. Algorithms for coloring reconfiguration under recolorability constraints. 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 37:1–37:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ISAAC.2018.37.

27 Michael J. Pelsmajer. Equitable list coloring and treewidth. *J. Graph Theory*, 61(2):127–139, 2009. doi:10.1002/jgt.20373.

28 Alexa Sharp. Distance coloring. In Lars Arge, Michael Hoffmann, and Emo Welzl, editors, *Algorithms – ESA 2007, 15th Annual European Symposium, Eilat, Israel, October 8-10, 2007, Proceedings*, volume 4698 of *Lecture Notes in Computer Science*, pages 510–521. Springer, 2007. doi:10.1007/978-3-540-75520-3_46.

29 Riste Skrekovski. Choosability of K_5-minor-free graphs. *Discret. Math.*, 190(1-3):223–226, 1998. doi:10.1016/S0012-365X(98)00158-7.

30 Michael D. Smith, Norman Ramsey, and Glenn H. Holloway. A generalized algorithm for graph-coloring register allocation. In William W. Pugh and Craig Chambers, editors, *Proceedings of the ACM SIGPLAN 2004 Conference on Programming Language Design and Implementation 2004, Washington, DC, USA, June 9-11, 2004*, pages 277–288. ACM, 2004. doi:10.1145/996841.996875.

31 Anthony Widjaja To. Parikh images of regular languages: Complexity and applications. *CoRR*, 2010. arXiv:1002.1464.

32 Margit Voigt. List colourings of planar graphs. *Discret. Math.*, 120(1-3):215–219, 1993. doi:10.1016/0012-365X(93)90579-I.

33 D.R. Woodall. *List colourings of graphs*, pages 269–301. London Mathematical Society Lecture Note Series. Cambridge University Press, 2001.

Enumerating the Irreducible Closed Sets of an Acyclic Implicational Base of Bounded Degree

Oscar Defrain
Aix-Marseille Université, CNRS, LIS, Marseille, France

Arthur Ohana
Aix-Marseille Université, CNRS, LIS, Marseille, France
Université Grenoble Alpes, Grenoble, France

Simon Vilmin
Aix-Marseille Université, CNRS, LIS, Marseille, France

Abstract

We consider the problem of enumerating the irreducible closed sets of a closure system given by an implicational base. To date, the complexity status of this problem is widely open, and it is further known to generalize the notorious hypergraph dualization problem, even in the case of acyclic convex geometries, i.e., closure systems admitting an acyclic implicational base. This paper studies this case with a focus on the degree, which corresponds to the maximal number of implications in which an element occurs. We show that the problem is tractable for bounded values of this parameter, even when relaxed to the notions of premise- and conclusion-degree. Our algorithms rely on a sequential approach leveraging from acyclicity, combined with the solution graph traversal technique for the case of premise-degree. They are shown to perform in incremental-polynomial time. These results are complemented in the long version of this document by showing that the dual problem of constructing the implicational base can be solved in polynomial time. Finally, we argue that our running times cannot be improved to polynomial delay using the standard framework of flashlight search.

2012 ACM Subject Classification Mathematics of computing → Enumeration

Keywords and phrases Algorithmic enumeration, closure systems, acyclic convex geometries, solution graph traversal, flashlight search, extension, hypergraph dualization

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.24

Related Version *Full Version*: `https://arxiv.org/abs/2506.24052` [16]

Funding The authors have been supported by the ANR project PARADUAL (ANR-24-CE48-0610-01).

Acknowledgements The authors are grateful to anonymous reviewers for their suggestions to improve this extended abstract as well as the long version.

1 Introduction

Finite closure systems appear in disguise in several areas of mathematics and computer science, including database theory [29, Chapter 13], Horn logic [12, Chapter 6], or lattice theory [13, Chapter 2], to mention but a few.

A closure system is a family of subsets of some finite ground set being closed by intersection and containing the ground set. The study of the various representations of closure systems and the algorithmic task of translating between them have received lots of attention in these respective fields, and have been the topic of the Dagstuhl Seminar 14201 [1]. Most importantly, the translation task we consider in this paper is known for being a considerable generalization of the notorious and widely open hypergraph dualization problem, while

© Oscar Defrain, Arthur Ohana, and Simon Vilmin;
licensed under Creative Commons License CC-BY 4.0

also not being known to be intractable, contrary to analogous other natural generalizations such as lattice dualization [4, 14, 26]. We refer the reader to the surveys [7, 33, 39] for more information on this topic.

Among the several representations of closure systems, two of them occupy a central role, namely, *irreducible closed sets* and *implicational bases*, whose formal definitions are postponed to Section 2. Informally, irreducible closed sets are elementary sets of the closure system from which every other can be recovered. On the other hand, implicational bases are sets of implications of the form $A \to B$, where A, the premise, and B, the conclusion, are subsets of the ground set. As noted in [28], these two representations are generally incomparable in size and usefulness. Notably, there exist instances where the minimum cardinality of an implicational bases is exponential in the number of irreducible closed sets, and vice versa. Moreover, the computational complexity of some algorithmic tasks such as reasoning, abduction, or recognizing lattice properties can differ significantly depending on the representation at hand [3, 25, 29]. These observations motivate the problem of translating between the two representations, in which one representation is given, and the other is to be computed. In this extended abstract, a focus is made on the problem of enumerating irreducible closed sets, denoted by ICS·Enum; the dual problem of constructing the implicational base is only addressed in the long version of this work.

Because of the possible exponential gap between the sizes of the two representations, input sensitive complexity is not a meaningful efficiency criterion when analyzing the performance of an algorithm solving ICS·Enum. Instead, the framework of algorithmic enumeration in which an enumeration algorithm has to list without repetitions a set of solutions, is usually adopted. The complexity of an algorithm is then analyzed from the perspective of *output-sensitive complexity*, which estimates the running time of the algorithm using both input and output sizes. In that context, an algorithm is considered tractable when it runs in *output-polynomial* time, that is, if its execution time is bounded by a polynomial in the combined sizes of the input and the output. This notion can be further constrained to guarantee some regularity in the enumeration. Namely, we say that an algorithm runs in *incremental-polynomial time* if it outputs the i^{th} solution in a time which is bounded by a polynomial in the input size plus i. It is said to run with *polynomial delay* if the time spent before the first output, after the last output, and between consecutive outputs is bounded by a polynomial in the input size only. We refer the reader to [24, 35] for more details on enumeration complexity.

In this work, we investigate *acyclic convex geometries*, a particular class of closure systems. They are the convex geometries – closure systems with the anti-exchange property – that admit an implicational base with acyclic implication-graph. Acyclic convex geometries have been extensively studied [8, 15, 23, 34, 37]. Even in this class though, the problem ICS·Enum was shown in [15] to be harder than distributive lattices dualization, a generalization of hypergraph dualization. On the other hand, the authors in [15] show that if, in addition to acyclicity, the implication graph admits a rank function, then the problem can be solved using hypergraph dualization. Using this result, they derive a tractable algorithm for so-called ranked convex geometries admitting an implicational base of bounded premise size. It should be noted that, even under these restrictions, the problem generalizes the one of enumerating all maximal independent sets of a graph (in fact of hypergraphs of bounded edge size) a central and notorious problem in enumeration [24, 36]. These results have then been extended to a more general setting in [32]. Other related results in the literature include the work [5] which deals with k-meet-semidistributive lattices, and closure systems of bounded Caratheodory number, see e.g., [39].

In this paper, we continue the line of research of characterizing tractable cases of acyclic convex geometries, by identifying for which parameters k one can derive tractable algorithms for ICS-Enum for fixed values of k. We note that in this research, finding an algorithm running in quasi-polynomial time should already be considered a success, since no sub-exponential time algorithms are known to date. We focus on the degree, which is defined as the maximum number of implications in which an element participates, and its two obvious relaxations which are the premise- and conclusion-degree; these parameters are defined in Section 2. The principal message of this work is that bounding one of these two relaxed notions of degree suffices to provide not only quasi-polynomial but polynomial algorithms for the translation. More specifically, as the first of our two main theorems, we obtain the following, whose quantitative formulation is postponed to Sections 4 and 5.

▶ **Theorem 1.** *There is an incremental-polynomial time algorithm enumerating the irreducible closed sets of an acyclic convex geometry given by an acyclic implicational base of bounded premise- or conclusion-degree.*

Our algorithms for acyclic convex geometries of bounded degree rely on several steps which combine structure and enumeration techniques. We give a rough description which highlight the techniques used, and refer to the appropriate sections for more details.

First, we use the fact that each irreducible closed set is *attached* to an element of the ground set, and that this element is unique in convex geometries; this is detailed in Section 2. Then, leveraging from acyclicity, our algorithm relies on a sequential procedure which consists of enumerating, incrementally and in a top-down fashion according to a topological ordering $x_1, \dots, x_n$ of the elements of the ground set, all the irreducible closed sets attached to x_i. This step can be seen as enumerating the irreducible closed set attached to x_i with additional information as an input, namely, the irreducible closed sets of ancestor elements. Finally, it remains to solve this later task, which we can do either by brute forcing minimal transversals of a carefully designed hypergraph in case of bounded conclusion-degree, or performing a solution graph traversal in case of bounded premise-degree. In the end, the resulting algorithms run in incremental-polynomial time, where the dependence in the number of solutions principally arises from the sequential top-down procedure.

Let us note that Theorem 1 also has implications on the dual problem of constructing an implicational base of minimum cardinality. Namely it allows us to obtain the following result whose description is detailed in the long version of this work [16] and included here for the reader familiar with closure systems.

▶ **Theorem 2.** *There is a polynomial-time algorithm constructing a minimum implicational base of an acyclic convex geometry of bounded premise- or conclusion-degree given by its irreducible closed sets.*

We note that in this direction, our algorithm is (input) polynomial thanks to the fact that the output has polynomial cardinality because of the bounded degree.

The remainder of the paper is organized as follows. Section 2 introduces the concepts and notations used throughout the paper. Section 3 presents the top-down approach that we use as the key ingredient in our algorithms. Sections 4 and 5 are devoted to proving Theorem 1 in the respective cases of bounded conclusion- and premise-degree. In Section 6 we discuss possible generalizations and open problems related to the current work, and also argue that our running times cannot be improved to polynomial delay using the standard framework of flashlight search.

Due to space constraints, the proofs of our different statements are not included in the present extended abstract: the interested reader is referred to [16] for more details.

2 Preliminaries

All the objects we consider in this paper are finite. If X is a set, $\mathcal{P}(X)$ is its powerset. The size of X is denoted $|X|$. A *set system*, or *hypergraph*, over a ground set X is a pair $(X, \mathcal{F})$ where $\mathcal{F}$ is a (non-empty) collection of subsets of X, that is $\mathcal{F} \subseteq \mathcal{P}(X)$. Its *size* is $|X| + |\mathcal{F}|$; note that the binary length of any standard encoding of a hypergraph is polynomially related to its size. Thus, for simplicity, we shall consider this measure for families as input sizes in the different problems we will consider. If X is clear from the context, we may identify $\mathcal{F}$ with $(X, \mathcal{F})$. In examples, if there is no ambiguity, we may write a set as the concatenation of its elements, for instance 123 instead of $\{1, 2, 3\}$. Similarly, to avoid cumbersome notations, we sometimes withdraw brackets when dealing with singletons, e.g., we may write say $\phi(x)$ instead of $\phi(\{x\})$.

Independent and hitting sets, hypergraph parameters. Let $\mathcal{H} = (X, \mathcal{E})$ be a hypergraph. We call *vertices* of $\mathcal{H}$ the elements in X, and *edges* the sets in $\mathcal{E}$. We say that $\mathcal{H}$ is *Sperner* if any two of its edges are pairwise incomparable by inclusion. An *independent set* of $\mathcal{H}$ is a set $I \subseteq X$ that contains no edges of $\mathcal{H}$, i.e., $E \nsubseteq I$ for every E in $\mathcal{E}$. The family of inclusion-wise maximal independent sets of $\mathcal{H}$ is denoted $\mathrm{MIS}(\mathcal{H})$. Formally, $\mathrm{MIS}(\mathcal{H}) = \max_{\subseteq}\{I : I \subseteq X,\ E \nsubseteq I \text{ for every } E \text{ in } \mathcal{E}\}$. Dually, a *hitting set*, or *transversal*, of $\mathcal{H}$ is a subset T of X that intersects each edge of $\mathcal{H}$. The collection of inclusion-wise minimal hitting sets of $\mathcal{H}$ is $\mathrm{MHS}(\mathcal{H}) = \min_{\subseteq}\{T : T \subseteq X,\ T \cap E \neq \emptyset \text{ for every } E \text{ in } \mathcal{E}\}$. The hypergraph $\mathrm{MHS}(\mathcal{H})$ is usually called the *dual hypergraph* of $\mathcal{H}$. Observe that $I \subseteq X$ is independent if and only if $T := X \setminus I$ is a hitting set. Consequently, $\mathrm{MHS}(\mathcal{H}) = \{X \setminus I : I \in \mathrm{MIS}(\mathcal{H})\}$. Moreover, if $\mathcal{H}$ is not Sperner, then its minimal hitting sets (resp. maximal independent sets) are precisely those of the hypergraph formed by the inclusion-wise minimal edges of $\mathcal{H}$. The two equivalent generation problems associated to maximal independent sets and minimal hitting sets read as follows:

Maximal Independent Sets Enumeration (MIS·Enum)
Input: A (Sperner) hypergraph $\mathcal{H} = (X, \mathcal{E})$.
Output: the family $\mathrm{MIS}(\mathcal{H})$.

Minimal Hitting Sets Enumeration (MHS·Enum)
Input: A (Sperner) hypergraph $\mathcal{H} = (X, \mathcal{E})$.
Output: the family $\mathrm{MHS}(\mathcal{H})$.

To date, the best algorithm for these tasks runs in output-quasi-polynomial time and is due to Fredman and Khachiyan [21]. However, a number of output-polynomial time algorithms have been proposed for particular cases, especially when bounding structural parameters [19], in particular the *dimension* of $\mathcal{H}$ being the size of its largest edge, and the *dual dimension*, being the dimension of the dual hypergraph of $\mathcal{H}$.

Closure systems, closure spaces. A *closure system* is a set system $(X, \mathcal{C})$ that satisfies $X \in \mathcal{C}$, and $C \cap C' \in \mathcal{C}$ whenever $C, C' \in \mathcal{C}$. The sets in $\mathcal{C}$ are called *closed (sets)*. A *closure space* is a pair (X, ϕ) where $\phi\colon \mathcal{P}(X) \to \mathcal{P}(X)$ is a mapping that satisfies for every $A, B \subseteq X$: (1) $A \subseteq \phi(A)$, (2) $\phi(A) \subseteq \phi(B)$ if $A \subseteq B$, and (3) $\phi(\phi(A)) = \phi(A)$. The map ϕ is a *closure operator*. Closure spaces and closure systems are in one-to-one correspondence. More precisely, if (X, ϕ) is a closure space, the family $\mathcal{C} = \{C : C \subseteq X,\ \phi(C) = C\}$ defines a closure system. Dually, if $(X, \mathcal{C})$ is a closure system, the map ϕ defined by $\phi(A) = \min_{\subseteq}\{C : C \in \mathcal{C},\ A \subseteq C\} = \bigcap\{C\colon C \in \mathcal{C},\ A \subseteq C\}$ is a closure operator whose

associated closure system is precisely $(X, \mathcal{C})$. In this paper, if (X, ϕ) is a closure space, we call $(X, \mathcal{C})$ the corresponding closure system and vice-versa. Moreover, we assume without loss of generality that $\emptyset \in \mathcal{C}$, a standard assumption.

Let $(X, \mathcal{C})$ be a closure system. The poset $(\mathcal{C}, \subseteq)$ of closed sets ordered by inclusion is a *(closure) lattice* where $C \wedge C' = C \cap C'$ and $C \vee C' = \phi(C \cup C')$. Let $M \in C$, $M \neq X$. The closed set M is *irreducible* if it is not the intersection of other distinct closed sets, that is, $M = C \cap C'$ for some $C, C' \in \mathcal{C}$ only if $C = M$ or $C' = M$. We denote by $\mathrm{irr}(\mathcal{C})$ the family of irreducible closed sets of $(X, \mathcal{C})$. For every $C \in \mathcal{C}$, we have $C = \bigcap\{M : M \in \mathrm{irr}(\mathcal{C}),\ C \subseteq M\}$. Moreover, $\mathrm{irr}(\mathcal{C})$ is the unique minimal subset of $\mathcal{C}$ from which $\mathcal{C}$ can be reconstructed using intersections. Let $x \in X$. We denote by $\mathrm{irr}(x)$ the inclusion-wise maximal closed sets not containing x, i.e., $\mathrm{irr}(x) = \max_{\subseteq}\{M : M \in \mathcal{C}, x \notin M\}$. Observe that $\mathrm{irr}(x) \subseteq \mathrm{irr}(\mathcal{C})$ for every $x \in X$. An irreducible closed set M in $\mathrm{irr}(x)$ is said to be *attached* to x. In fact, it is well known that $\mathrm{irr}(\mathcal{C}) = \bigcup_{x \in X} \mathrm{irr}(x)$ (see, e.g., [29, 38]) and this union will even be disjoint for us (see Proposition 3). An example of a closure system and its irreducible closed sets is depicted in Figure 1.

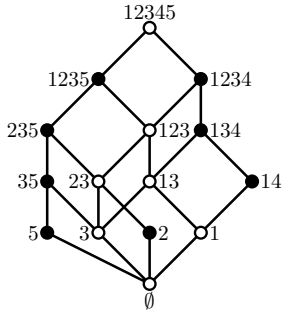


Figure 1 A closure system over $X = \{1, 2, 3, 4, 5\}$ represented via the Hasse diagram of its closure lattice. Black dots are irreducible closed sets which can be seen by the fact that they have a unique successor. We have, for instance, $\phi(25) = 235$ and $\phi(4) = 14$, $\mathrm{irr}(3) = \{5, 2, 14\}$ and $\mathrm{mingen}(3) = \{25, 15, 24, 45\}$. An implicational base for this closure system is for instance $\Sigma = \{4 \to 1, 25 \to 3, 12 \to 3, 15 \to 23\}$.

Implicational bases. An *implication* over X is a statement $A \to B$ where A, B are subsets of X. In $A \to B$, A is the *premise* and B the *conclusion*. An *implicational base* (IB) is a pair (X, Σ) where Σ is a collection of implications over X. Given an IB (X, Σ), $|\Sigma|$ is the cardinality of Σ, that is the number of implications in Σ. The *size* of the IB is the value $|X| + |\Sigma|$. The *degree* of an element $x \in X$, denoted $\deg(x)$, is the number of implications of Σ in which x appears. We refine this notion by defining the *premise-degree* of x as $\mathrm{pdeg}(x) := |\{A : A \to B \in \Sigma, x \in A\}|$, and its *conclusion-degree* as $\mathrm{cdeg}(x) := |\{A \to B : A \to B \in \Sigma, x \in B\}|$. Then $\deg(x) \leq \mathrm{pdeg}(x) + \mathrm{cdeg}(x)$. Equality may not hold in general because an element can belong to both premise and the conclusion of an implication. In this paper though, we avoid this degenerate case by always considering the premise and conclusion of an implication are disjoint. Moreover, we will mainly use the *degree*, *premise-degree*, and *conclusion-degree* of Σ defined by $\mathrm{pdeg}(\Sigma) := \max_{x \in X} \mathrm{pdeg}(x)$, $\mathrm{cdeg}(\Sigma) := \max_{x \in X} \mathrm{cdeg}(x)$, $\deg(\Sigma) := \max_{x \in X} \deg(x)$. Note that $\deg(\Sigma) \leq \mathrm{pdeg}(\Sigma) + \mathrm{cdeg}(\Sigma)$ for any Σ. As an example, for the implicational base given in Figure 1, we have $\mathrm{pdeg}(2) = 2$ and $\mathrm{cdeg}(2) = 1$.

We now relate IBs to closure systems. Let (X, Σ) be an IB. A set $C \subseteq X$ is *closed* for (X, Σ) if for every implication $A \to B \in \Sigma$, $A \subseteq C$ implies $B \subseteq C$. The pair $(X, \mathcal{C})$ where $\mathcal{C}$ is the family of closed sets of (X, Σ) is a closure system. An IB (X, Σ) thus induces a

closure operator $\phi(\cdot)$ that can be computed in polynomial time using the *forward chaining* algorithm. This procedure starts from a set Y and produces a sequence of sets $Y = Y_0 \subseteq Y_1 \subseteq \cdots \subseteq Y_k = \phi(Y)$ such that $Y_i = Y_{i-1} \cup \bigcup\{B : A \to B \in \Sigma, A \subseteq Y_{i-1}, B \nsubseteq Y_{i-1}\}$ for $1 \leq i \leq k$. On the other hand, any closure system $(X, \mathcal{C})$ can be represented by several *equivalent* IBs. For instance, $\Sigma = \{A \to \phi(A) : A \subseteq X\}$ constitutes an IB for $(X, \mathcal{C})$ as well as $\Sigma' = \{C \cup \{x\} \to \phi(C \cup \{x\}) : C \in \mathcal{C}, x \notin C\}$ (recall that we assume $\emptyset \in \mathcal{C}$). If (X, Σ) is an IB, we use the notation $\mathrm{irr}(\Sigma)$ to denote the irreducible closed sets of the associated closure system. An implicational base for the closure system of Figure 1 is given in Figure 2.

The *degree* of a closure system is the least integer k such that it admits an implicational bases of degree k. The premise- and conclusion-degree are defined analogously.

Convex geometries, acyclic convex geometries. A closure system $(X, \mathcal{C})$ is a *convex geometry* if for every closed set C distinct from X there exists $x \notin C$ such that $C \cup \{x\}$ is closed. Equivalently, $(X, \mathcal{C})$ is a convex geometry if the corresponding closure operator ϕ has the *anti-exchange* property: for every closed set C and each distinct $x, y \notin C$, $x \in \phi(C \cup \{y\})$ entails $y \notin \phi(C \cup \{x\})$. The closure system depicted in Figure 1 is a convex geometry. Convex geometries have been rediscovered in several fields and, as such, enjoy several characterizations [31]. In particular, they can be characterized by their irreducible closed sets, as recalled in the subsequent statement:

▶ **Proposition 3** (see e.g., [18, Theorem 2.4]). *A closure system $(X, \mathcal{C})$ is a convex geometry if and only if each irreducible closed set is attached to a unique element, i.e., $\{\mathrm{irr}(x) : x \in X\}$ is a partition of $\mathrm{irr}(\mathcal{C})$.*

Let $T \subseteq X$. We call *irreducible selection* for T a family of sets obtained by choosing, for each $t \in T$, one $M \in \mathrm{irr}(t)$. Note that since we are dealing with convex geometries, and by Proposition 3, the closed sets M are all distinct and the resulting family has the same cardinal as T. In the following, we denote by $\mathcal{S}(T)$ the family of all irreducible selections for a given set T. We will use irreducible selections as a way to forbid elements, as stated in the following observation.

▶ **Observation 4.** *For any $T \subseteq X$ and any selection $\mathcal{S} \in \mathcal{S}(T)$, the intersection $\bigcap \mathcal{S}$ of all sets in $\mathcal{S}$ is disjoint from T, i.e., it contains no $t \in T$.*

Let us now turn our attention to acyclic convex geometries. They have been studied under various names such as poset-type convex geometries [34], G-geometries [37], or as acyclic Horn functions from the perspective of Horn CNFs [8, 23]. Let (X, Σ) be an IB. The *implication-graph* [8, 23] of (X, Σ) is the directed graph $G(\Sigma) = (X, E)$ where there is an arc ab in E if there exists $A \to B$ in Σ such that $a \in A$ and $b \in B$. The IB (X, Σ) is *acyclic* if $G(\Sigma)$ is. In this case, $G(\Sigma)$ can be seen as a poset, and naturally induces an *ancestor* relation over X defined for all $x \in X$ as follows:

$$\mathrm{anc}(x) = \{y : y \in X,\ x \neq y, \text{ and there exists a directed path from } y \text{ to } x \text{ in } G(\Sigma)\}$$

Equivalently, $\mathrm{anc}(x)$ is the set of in-neighbors of x in the transitive closure of $G(\Sigma)$. An element x of X is thus maximal if it has no ancestors, and minimal if it has no descendants. As an example, Figure 2 illustrates the implication-graph of the IB given in Figure 1. A closure system that admits an acyclic IB is *acyclic* and it is a well-known fact that acyclic closure systems are convex geometries [34, 37].

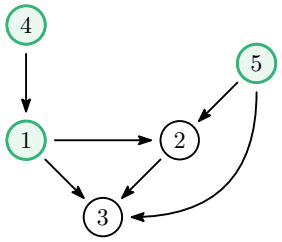


Figure 2 The implication-graph $G(\Sigma)$ of the IB (X, Σ) of Figure 1. This directed graph is acyclic, which makes the closure system an acyclic convex geometry. The ancestors of 2, highlighted in green, are $1, 4, 5$, i.e., $\text{anc}(2) = \{1, 4, 5\}$. Maximal elements are 4, 5 and the unique minimal element is 3. Here, an example of *topological order* that we will refer to in the next section is $4, 1, 5, 2, 3$.

Irreducible closed sets enumeration. To conclude the preliminaries, we formally present the enumeration problem we study in this paper and review its main aspects.

Irreducible Closed Sets Enumeration (ICS·Enum)
Input: An implicational base (X, Σ).
Output: The irreducible closed sets of the associated closure system.

Let us stress the fact that in ICS·Enum, only an implicit (and possibly compact) description of the closure system is provided, namely, the implicational base. Typically, the output of ICS·Enum may be of exponential size with respect to the input size, as illustrated in Example 5 below. Hence, the problem is studied through the lens of output-sensitive complexity.

▶ **Example 5.** Let $X = \{a_1, \dots, a_n\} \cup \{b_1, \dots, b_n\} \cup \{x\}$, and $\Sigma = \{a_i b_i \to x : 1 \leq i \leq n\}$. Denoting $(X, \mathcal{C})$ the associated closure system, we have $\text{irr}(\mathcal{C}) = \{X \setminus \{y\} : y \in X,\ y \neq x\} \cup \{\{c_1, \dots, c_n\} : c_i \in \{a_i, b_i\},\ 1 \leq i \leq n\}$. Therefore, $|\text{irr}(\mathcal{C})| = 2^n + 2n$ while $|\Sigma| = n$.

As explained in the introduction, this problem has been studied in different fields [3,17,22,28,38] and its complexity remains unsettled to date. Yet it is harder than MIS·Enum, in the sense that an output-polynomial time algorithm for one of ICS·Enum implies one for MIS·Enum. Moreover, ICS·Enum reduces to the problem of enumerating the irreducible closed sets attached to a point, which we call ACS·Enum, and that will be considered in this paper:

Attached Irreducible Closed Set Enumeration (ACS·Enum)
Input: An implicational base (X, Σ) and $x \in X$.
Output: The family $\text{irr}(x)$ of irreducible closed sets attached to x.

Indeed, an algorithm solving ACS·Enum in output-polynomial time can be applied to each element x to produce all irreducible closed sets of a closure system, with at most a linear number of repetitions in general, and no repetitions in convex geometries by Proposition 3. Unfortunately, it was shown by Kavvadias et al. [26] in the language of Horn CNFs that the problem ACS·Enum admits no output-polynomial time algorithm unless $\mathsf{P} = \mathsf{NP}$.

Nevertheless, in the remaining of the paper, we will show that a variation of the problem taking profit of the acyclicity of the input allows us to obtain tractable algorithms for acyclic convex geometries of bounded degree.

3 Top-down procedure

We describe the general sequential procedure that we use to solve ICS·Enum. We use a similar approach to tackle the dual problem of constructing the implicational base in the long version of this work [16]. Intuitively, it consists in enumerating, incrementally and in a top-down fashion according to a topological ordering $x_1, \ldots, x_n$ of the implicational base, all the irreducible closed sets attached to x_i.

Recall that the ancestors of an element x in an acyclic implicational base are the elements that participate in a path of implications to x, and which can be identified in polynomial time; see Section 2. More formally, we show that ICS·Enum reduces to the following version of ACS·Enum in which we are given ancestors' irreducible closed sets as additional information.

Attached Closed Sets Enumeration with Ancestors' solutions (ACS+A·Enum)
Input: An acyclic implicational base (X, Σ), an element $x \in X$, and the family $\mathrm{irr}(y)$ for every ancestor y of x in Σ.
Output: The set $\mathrm{irr}(x)$.

Clearly, an algorithm for ACS·Enum can be used to solve ACS+A·Enum, while the reverse may not be true. We argue that by iteratively using an algorithm for ACS+A·Enum on a maximal element of the implicational base among those that have not been selected yet, we obtain an algorithm for ICS·Enum. Note that the base case corresponds to computing the irreducible closed set of maximal elements x verifying $\mathrm{irr}(x) = \{X \setminus \{x\}\}$. Moreover, this procedures preserves incremental-polynomial bounds. In terms of difficulty, ACS+A·Enum may thus be seen as standing as an intermediate problem between ACS·Enum and ICS·Enum in acyclic convex geometries. More formally, we obtain the following.

▶ **Theorem 6.** *Let $f\colon \mathbb{N}^2 \to \mathbb{N}$ be a function. Let us assume that there is an algorithm that, given an instance of ACS+A·Enum of size N, produces its i^{th} solution in $f(N, i)$-time. Then there is an algorithm that, given an acyclic instance of ICS·Enum of size N', produces its j^{th} solution in $O(\mathrm{poly}(N') + f(N' + j, j))$ time.*

We note that an analogous statement to the one in Theorem 6 holds by relaxing the required and obtained time bounds to output-polynomial times. On the other hand, improving the complexity of an algorithm for ACS+A·Enum to polynomial delay (or even input-polynomial as we will provide later for the case of bounded conclusion-degree) does not generally ensure polynomial delay (or incremental-linear) for ICS·Enum, since the size of an instance of ACS+A·Enum grows with the number of already obtained solutions for ICS·Enum, which typically becomes exponential in the size of the implicational base.

Further limitations related to the top-down approach – in particular on the hardness of ACS·Enum on implicational bases of height at most two – are discussed in the long version of this work [16]. Nevertheless, we shall argue in the remaining of the paper that this approach yields tractable algorithms when bounding structural parameters such as the premise- and conclusion-degree.

4 Bounded conclusion-degree

We show that the top-down approach successfully provides an incremental-polynomial time algorithm for ICS·Enum whenever the implicational base has bounded conclusion-degree, proving one of the two cases of Theorem 1.

In the following, let us consider an instance of ACS+A·Enum, i.e., we fix some acyclic implicational base (X, Σ), an element $x \in X$, and assume that we are given $\mathrm{irr}(y)$ for all ancestors y of x in Σ. Let us consider the hypergraph defined by

$$\mathcal{E}_x := \{A : \ A \to B \in \Sigma, \ x \in B\}, \text{ and}$$
$$\mathcal{H}_x := \left(\bigcup \mathcal{E}_x, \mathcal{E}_x\right).$$

The intuition behind the following lemma is that, in order to get an irreducible closed set attached to x, one should remove at least one element a in the premise of each implication having x in the conclusion, which can be done by considering the intersection with an irreducible closed set attached to a. Recall that $\mathcal{S}(T)$ denotes the family of irreducible selections; see Section 2.

▶ **Lemma 7.** *For every $M \in \mathrm{irr}(x)$ there exists a minimal transversal T of $\mathcal{H}_x$ and an irreducible selection $\mathcal{S} \in \mathcal{S}(T)$ such that $M = (\bigcap \mathcal{S}) \setminus \{x\}$.*

Note that, by definition, the number of edges in $\mathcal{H}_x$ is exactly $\mathrm{cdeg}(x)$, which is at most $\mathrm{cdeg}(\Sigma)$. Moreover, since the ground set of $\mathcal{H}_x$ is a subset of the ancestors of x, the selections of Lemma 7 are part of the input we consider for ACS+A·Enum. By brute-forcing the irreducible selections for every transversal, we derive a polynomial-time algorithm for ACS+A·Enum whenever the conclusion-degree of x is bounded by a constant.

▶ **Theorem 8.** *There is an algorithm solving ACS+A·Enum in $N^{O(k)}$ time on inputs of size N and maximum conclusion-degree k.*

We conclude to the following, as a corollary of Theorems 6 and 8, and proving the case of bounded-conclusion degree in Theorem 1.

▶ **Theorem 9.** *There is an incremental-polynomial time algorithm solving ICS·Enum on acyclic implicational bases of bounded conclusion-degree.*

Let us remark that in our algorithm, we actually do not need the knowledge of $\mathrm{irr}(a)$ for *all* ancestors of x, but only for those in a premise which has x in conclusion. This however makes no asymptotic difference on the final complexity we get for ICS·Enum.

Also, it appears in the proof of Theorem 8 that we only need a constant upper bound on the size k of a minimal transversal of $\mathcal{H}_x$, to derive an input-polynomial time algorithm solving ACS+A·Enum. We immediately derive the following generalization of Theorem 9 dealing with hypergraphs of bounded dual-dimension (admitting minimal transversals of bounded size) who define complements of conformal hypergraphs [6].

▶ **Theorem 10.** *There is an incremental-polynomial time algorithm solving ICS·Enum if, for any element x of the ground set, the hypergraph of premises having x in their conclusion has bounded dual-dimension.*

We end the section by showing the limitations of this approach in order to get output-polynomial time bounds whenever we drop the condition of bounded conclusion-degree.

First, note that the key step of the approach is to compute all possible intersections as provided in Lemma 7. However, there are cases where the number of possible such intersections is exponential in $|\mathrm{irr}(\Sigma)|$, which typically occurs if minimal transversals of $\mathcal{H}_x$ are of unbounded size, or if $\mathrm{MHS}(\mathcal{H}_x)$ is of exponential size in $\mathrm{irr}(x)$.

To get one such example, consider $k \in \mathbb{N}$ and the implicational base defined by $X = \{a_1, b_1, \ldots, a_k, b_k, x\}$ and $\Sigma = \{a_i b_i \to x : 1 \leq i \leq k\} \cup \{b_i \to b_{i+1} : 1 \leq i < k\}$; see Figure 3 for an illustration. Then $\mathrm{irr}(x) = \{(a_i)_{1 \leq i \leq j} \cup (b_i)_{j+1 \leq i \leq k} : 0 \leq j \leq k\}$ so $|\mathrm{irr}(x)| = k+1$, and

the other elements in $X \setminus \{x\}$ are easily seen to have a single attached closed set. Therefore, $\mathrm{irr}(\Sigma)$ is of polynomial size in $|X| = 2k + 1$ and $|\Sigma| = 2k - 1$. However, $|\mathrm{MHS}(\mathcal{H}_x)| = 2^k$ and computing this set would fail in providing an output-polynomial time algorithm for enumerating $\mathrm{irr}(x)$.

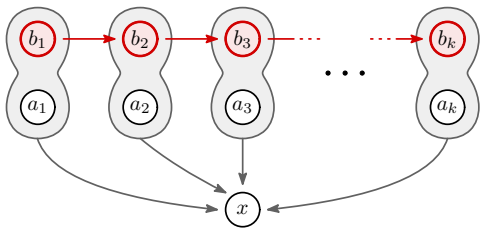


Figure 3 An instance where $|\mathrm{MHS}(\mathcal{H}_x)| = 2^k$ but $|\mathrm{irr}(\Sigma)| = 3k + 1$ and $|\Sigma| = 2k - 1$.

At last, we note that this example has bounded premise-degree. We will show however in the next section that such instances can actually be efficiently solved by solution graph traversal.

5 Bounded premise-degree

We provide an incremental-polynomial time algorithm solving ICS·Enum whenever the given implicational base has bounded premise-degree. Our algorithm relies on the top-down approach from Section 3 and solves ACS+A·Enum relying on the solution graph traversal technique.

In the *solution graph traversal* framework, one has to define a digraph called *solution graph*, whose nodes are the solutions we aim to enumerate, and whose arcs (F, F') consists of ways of reconfigurating a given solution F into another solution F'. Then, the goal is not to build the solution graph entirely, but rather to traverse all its nodes efficiently and avoiding repetitions of nodes. This general technique has been fruitfully applied to numerous enumeration problems, see e.g., [9, 27, 40], and generalized to powerful frameworks [2, 10, 11]. In the following we will use the next folklore result which states conditions for such a technique to be efficient; see [20] for a quantitative statement and a proof.

▶ **Theorem 11** (Folklore). *Let $\mathcal{F} \subseteq \mathcal{P}(X)$ be a solution set over a ground set X, and let us denote by n the size of the input. Then there is a* $\mathrm{poly}(n)$*-delay algorithm enumerating $\mathcal{F}$ whenever:*

1. *one member of $\mathcal{F}$ can be found in* $\mathrm{poly}(n)$*-time;*
2. *there is a function $\mathcal{N} : \mathcal{F} \to \mathcal{P}(\mathcal{F})$ called* neighboring function *mapping every solution to a set of solutions that can be computed in* $\mathrm{poly}(n)$ *time given $F \in \mathcal{F}$;*
3. *the directed solution graph $(\mathcal{F}, \{(F, F') : F' \in \mathcal{N}(F)\})$ is strongly connected.*

In the following, let us consider an instance of ACS+A·Enum, i.e., we fix some acyclic implicational base (X, Σ), an element $x \in X$, and we assume that we are given $\mathrm{irr}(y)$ for all ancestors y of x.

We first argue that Condition 1 holds for ACS+A·Enum. Let GC_x be a function which maps any $E \subseteq X$ such that $x \notin \phi(E)$ to a maximal set $M \supseteq E$ with this property. We call the resulting set the *greedy completion* of E. Note that $\mathrm{GC}_x(E)$ can be computed in polynomial time in the size of (X, Σ), and that the obtained set lies in $\mathrm{irr}(x)$; see Section 2. We derive the following.

▶ **Lemma 12.** *A first solution* $\mathrm{GC}_x(\emptyset) \in \mathrm{irr}(x)$ *can be obtained in* $\mathrm{poly}(|X| + |\Sigma|)$ *time.*

We now turn on showing the existence of a polynomial-time computable neighboring function satisfying Conditions 2 and 3. Given some irreducible closed set $M \in \mathrm{irr}(x)$, and some element $y \in X \setminus M$, we define

$$\mathcal{E}_{M,y} := \{A : A \to B \in \Sigma \text{ with } A \setminus M = \{y\},\ B \not\subseteq M\},$$
$$V_{M,y} := \bigcup \mathcal{E}_{M,y} \text{ and } \mathcal{H}_{M,y} := (V_{M,y}, \mathcal{E}_{M,y}).$$

In other words, $\mathcal{H}_{M,y}$ is the hypergraph of premises of Σ not included in M, triggered by adding y, and with a conclusion not in M. Note that $|\mathcal{E}_{M,y}| \leq \mathrm{pdeg}(y) \leq \mathrm{pdeg}(\Sigma)$. Also, because every premise in $\mathcal{E}_{M,y}$ implies a $b \notin M$, hence an ancestor of x, we derive the following observation by transitivity.

▶ **Observation 13.** *Every vertex in* $V_{M,y}$ *is an ancestor of* x*.*

Regarding Condition 2, we first need the following statement, which holds the same idea as Lemma 7.

▶ **Lemma 14.** *Let* M, M^* *be two distinct sets in* $\mathrm{irr}(x)$*. Then there exist* $y \in M^* \setminus M$*,* $T \in \mathrm{MHS}(\mathcal{H}_{M,y})$*, and* $\mathcal{S} \in \mathcal{S}(T)$ *such that* $M^* \subseteq \bigcap \mathcal{S}$*.*

We are now ready to define the aforementioned neighboring function. Given $M \in \mathrm{irr}(x)$ and $y \notin M$, we define

$$\mathcal{N}(M, y) := \left\{ \mathrm{GC}_x\left(\left(M \cap \bigcap \mathcal{S}\right) \cup \{y\}\right) : \begin{array}{l} \mathcal{S} \in \mathcal{S}(T) \text{ and } T \in \mathrm{MHS}(\mathcal{H}_{M,y}) \\ \text{such that } x \notin \phi\big((M \cap \bigcap \mathcal{S}) \cup \{y\}\big) \end{array} \right\}$$

and put $\mathcal{N}(M) := \bigcup_{y \notin M} \mathcal{N}(M, y)$. We prove that Conditions 2 and 3 hold for this function.

▶ **Observation 15.** *By definition, the greedy completion in the definition of* $\mathcal{N}(M, y)$ *is well defined and* $\mathcal{N}(M) \subseteq \mathrm{irr}(x)$*.*

▶ **Lemma 16.** *Let* N *be the sum of the sizes of* X*,* Σ*, and* $\mathrm{irr}(a)$ *for all* $a \in \mathrm{anc}(x)$*. Then, for all* $M \in \mathrm{irr}(x)$*,* $\mathcal{N}(M)$ *can be computed in* $N^{O(k)}$ *time where* k *is the premise-degree of* Σ*.*

Note that Condition 2 is ensured by Observation 15 and Lemma 16. We are thus left to prove Condition 3, which we obtain by proving that for any pair of solutions, there is a neighbor of the first one with greater intersection with the second.

▶ **Lemma 17.** *Let* M, M^* *be two distinct sets in* $\mathrm{irr}(x)$*. Then there exists* $M' \in \mathcal{N}(M)$ *such that* $M \cap M^* \subsetneq M' \cap M^*$*.*

▶ **Corollary 18.** *The solution graph* $(\mathrm{irr}(x), \{(M, M') : M' \in \mathcal{N}(M)\})$ *is strongly connected.*

Combining Theorem 11, Lemma 12, Lemma 16, and Corollary 18, we obtain a polynomial-delay algorithm for ACS+A·Enum by solution graph traversal for acyclic implicational bases of bounded premise-degree. This, together with Theorem 6, in turn implies the following theorem, proving the remaining case of bounded premise-degree in Theorem 1.

▶ **Theorem 19.** *There is an incremental-polynomial time algorithm solving ICS·Enum on acyclic implicational bases of bounded premise-degree.*

As in the previous section, we only needed to bound the size of a minimal transversal of $\mathcal{H}_{M,y}$ in the proof of previous theorem to derive an input-polynomial time algorithm solving ACS+A·Enum. This gives the following result, analogous to Theorem 10.

▶ **Theorem 20.** *There is an incremental-polynomial time algorithm solving ICS·Enum if, for any irreducible closed set* M *and* $y \notin M$*, the hypergraph* $\mathcal{H}_{M,y}$ *has bounded dual-dimension.*

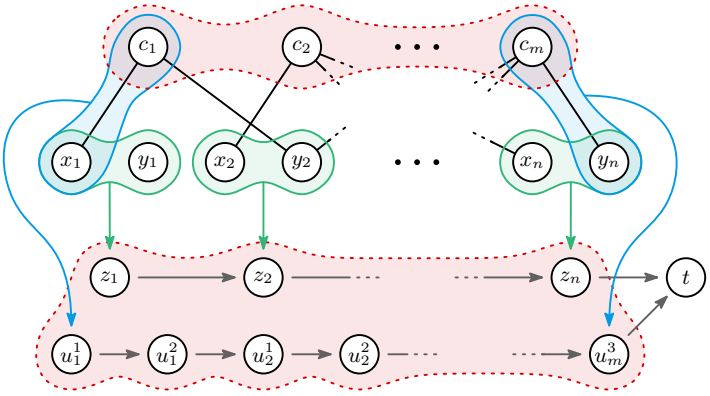


Figure 4 An illustration of the construction from Theorem 21. Black edges correspond to the incidence bipartite graph of a given CNF with clauses' size at most 3, and variable occurring in at most 3 clauses. There is a blue implication for each such edge (partially represented). The green implications represent conflicting literals. With this reduction, there is an irreducible closed set attached to t and avoiding the red set if and only if the formula is satisfiable.

6 Conclusion

In this paper, we described two incremental-polynomial time algorithms for the enumeration of irreducible closed sets in acyclic convex geometries of bounded (premise- or conclusion-) degree. This solves one of the two translation task for this class, the other one being addressed in the long version of this work [16]. We note that the delay of our algorithms is intrinsically not polynomial since our algorithm rely on a sequential top-down procedure which reduces the enumeration to an auxiliary enumeration problem whose instance grows with the total number of solutions output so far. This suggests the following question.

▶ **Question 1.** *Can ICS·Enum be solved with polynomial delay for acyclic convex geometries of bounded degree?*

Toward this direction, we now argue that the classic flashlight search framework may not be used in a straightforward way to obtain a positive answer to this question. In the *flashlight search technique*, the goal is to construct solutions one element at a time, according to a linear order $x_1, \ldots, x_n$ on the ground set, and deciding at each step whether x_i should or should not be included in the solution. For this approach to be tractable, the framework requires to solve the so-called "extension problem" in polynomial time: given two disjoint subsets K, F of the ground set, decide whether there exists a solution including K and avoiding F. We refer to, e.g., [9, 30] for a more detailed description of this folklore technique. More formally, this technique reduces the enumeration to the following decision problem.

Irreducible Closed Set Extension (ICS·Ext)
Input: An implicational base (X, Σ) and $K, F \subseteq X$.
Question: Does there exist $M \in \mathrm{irr}(\mathcal{C})$ such that $K \subseteq M$ and $M \cap F = \emptyset$?

We show that this problem is NP-complete even for acyclic implicational bases of bounded degree with $K = \emptyset$, suggesting that this framework may not be used in a straightforward way to improve our results. Let us however remark that this does not rule out the possibility of a polynomial-delay algorithm for the problem, using a different approach or relying on particular vertex orderings. We refer to [16] for the proof of this result, the construction of which is hinted in Figure 4.

▶ **Theorem 21.** *The extension problem for ICS·Enum is* NP*-complete even for acyclic implicational bases with degree, premise-degree, conclusion-degree and dimension (i.e., the maximum size of a premise) simultaneously bounded by 4, 4, 2 and 2, respectively.*

Note that, as an intermediate step, one may wonder whether there exists an incremental-polynomial time algorithm for ICS·Enum where the degree of the polynomial in the number of solutions is independent of the premise degree.

Among other directions, it is natural to ask whether our results extend to broader classes of closure systems or to parameters that generalize the degree. We include a thorough discussion on the relevant classes and parameters in the long version of this work [16] as well as some considerations related to parameterized enumeration. In particular, we argue that without acyclicity the degree alone is not helpful anymore as ICS·Enum for instances with bounded degree becomes as general as ICS·Enum.

References

1 Kira V. Adaricheva, Giuseppe F. Italiano, Hans Kleine Büning, and György Turán. Horn formulas, directed hypergraphs, lattices and closure systems: related formalisms and applications (Dagstuhl Seminar 14201). *Dagstuhl Reports*, 4(5):1–26, 2014. doi:10.4230/DagRep.4.5.1.

2 David Avis and Komei Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65(1-3):21–46, 1996. doi:10.1016/0166-218X(95)00026-N.

3 Mikhail A. Babin and Sergei O. Kuznetsov. Computing premises of a minimal cover of functional dependencies is intractable. *Discrete Applied Mathematics*, 161(6):742–749, 2013. doi:10.1016/J.DAM.2012.10.026.

4 Mikhail A. Babin and Sergei O. Kuznetsov. Dualization in lattices given by ordered sets of irreducibles. *Theoretical Computer Science*, 658:316–326, 2017. doi:10.1016/J.TCS.2016.01.005.

5 Laurent Beaudou, Arnaud Mary, and Lhouari Nourine. Algorithms for k-meet-semidistributive lattices. *Theoretical Computer Science*, 658:391–398, 2017. doi:10.1016/J.TCS.2015.10.029.

6 Claude Berge. *Hypergraphs: combinatorics of finite sets*, volume 45. Elsevier, 1984.

7 Karell Bertet, Christophe Demko, Jean-François Viaud, and Clément Guérin. Lattices, closures systems and implication bases: a survey of structural aspects and algorithms. *Theoretical Computer Science*, 743:93–109, 2018. doi:10.1016/J.TCS.2016.11.021.

8 Endre Boros, Ondřej Čepek, Alexander Kogan, and Petr Kučera. A subclass of Horn CNFs optimally compressible in polynomial time. *Annals of Mathematics and Artificial Intelligence*, 57(3-4):249–291, 2009. doi:10.1007/S10472-010-9197-7.

9 Endre Boros, Khaled Elbassioni, and Vladimir Gurvich. Algorithms for generating minimal blockers of perfect matchings in bipartite graphs and related problems. In *European Symposium on Algorithms*, pages 122–133. Springer, 2004. doi:10.1007/978-3-540-30140-0_13.

10 Sara Cohen, Benny Kimelfeld, and Yehoshua Sagiv. Generating all maximal induced subgraphs for hereditary and connected-hereditary graph properties. *Journal of Computer and System Sciences*, 74(7):1147–1159, 2008. doi:10.1016/J.JCSS.2008.04.003.

11 Alessio Conte, Roberto Grossi, Andrea Marino, Takeaki Uno, and Luca Versari. Proximity search for maximal subgraph enumeration. *SIAM Journal on Computing*, 51(5):1580–1625, 2022. doi:10.1137/20M1375048.

12 Yves Crama and Peter L. Hammer. *Boolean functions: Theory, algorithms, and applications*. Cambridge University Press, 2011.

13 Brian A. Davey and Hilary A. Priestley. *Introduction to lattices and order*. Cambridge university press, 2002.

14 Oscar Defrain and Lhouari Nourine. Dualization in lattices given by implicational bases. *Theoretical Computer Science*, 814:169–176, 2020. doi:10.1016/J.TCS.2020.01.028.

15 Oscar Defrain, Lhouari Nourine, and Simon Vilmin. Translating between the representations of a ranked convex geometry. *Discrete Mathematics*, 344(7):112399, 2021. doi:10.1016/J.DISC.2021.112399.

16 Oscar Defrain, Arthur Ohana, and Simon Vilmin. Translating between the representations of an acyclic convex geometry of bounded degree. *arXiv preprint*, 2025. doi:10.48550/arXiv.2506.24052.

17 Jean-Paul Doignon and Jean-Claude Falmagne. *Knowledge spaces*. Springer Science & Business Media, 2012.

18 Paul H. Edelman and Robert E. Jamison. The theory of convex geometries. *Geometriae dedicata*, 19(3):247–270, 1985.

19 Thomas Eiter, Kazuhisa Makino, and Georg Gottlob. Computational aspects of monotone dualization: A brief survey. *Discrete Applied Mathematics*, 156(11):2035–2049, 2008. doi:10.1016/J.DAM.2007.04.017.

20 Khaled Elbassioni. A polynomial delay algorithm for generating connected induced subgraphs of a given cardinality. *Journal of Graph Algorithms and Applications*, 19(1):273–280, 2015. doi:10.7155/JGAA.00357.

21 Michael L. Fredman and Leonid Khachiyan. On the complexity of dualization of monotone disjunctive normal forms. *Journal of Algorithms*, 21(3):618–628, 1996. doi:10.1006/JAGM.1996.0062.

22 Georg Gottlob and Leonid O. Libkin. Investigations on Armstrong relations, dependency inference, and excluded functional dependencies. *Acta Cybernetica*, 9(4):385–402, 1990. URL: https://cyber.bibl.u-szeged.hu/index.php/actcybern/article/view/3383.

23 Peter L. Hammer and Alexander Kogan. Quasi-acyclic propositional Horn knowledge bases: Optimal compression. *IEEE Transactions on knowledge and data engineering*, 7(5):751–762, 1995. doi:10.1109/69.469822.

24 David S. Johnson, Mihalis Yannakakis, and Christos H. Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27(3):119–123, 1988. doi:10.1016/0020-0190(88)90065-8.

25 Henry A. Kautz, Michael J. Kearns, and Bart Selman. Reasoning with characteristic models. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, AAAI'93, pages 34–39. AAAI Press, 1993. URL: http://www.aaai.org/Library/AAAI/1993/aaai93-006.php.

26 Dimitris J. Kavvadias, Martha Sideri, and Elias C. Stavropoulos. Generating all maximal models of a Boolean expression. *Information Processing Letters*, 74(3-4):157–162, 2000. doi:10.1016/S0020-0190(00)00023-5.

27 Leonid Khachiyan, Endre Boros, Khaled Elbassioni, and Vladimir Gurvich. On enumerating minimal dicuts and strongly connected subgraphs. *Algorithmica*, 50(1):159–172, 2008. doi:10.1007/S00453-007-9074-X.

28 Roni Khardon. Translating between Horn representations and their characteristic models. *Journal of Artificial Intelligence Research*, 3:349–372, 1995. doi:10.1613/JAIR.183.

29 Heikki Mannila and Kari-Jouko Räihä. *The design of relational databases*. Addison-Wesley Longman Publishing Co., Inc., 1992.

30 Arnaud Mary and Yann Strozecki. Efficient enumeration of solutions produced by closure operations. *Discret. Math. Theor. Comput. Sci.*, 21(3), 2019. doi:10.23638/DMTCS-21-3-22.

31 Bernard Monjardet. A use for frequently rediscovering a concept. *Order*, 1(4):415–417, 1985.

32 Lhouari Nourine and Simon Vilmin. Hierarchical decompositions of implicational bases for the enumeration of meet-irreducible elements. *Theoretical Computer Science*, 969:114030, 2023. doi:10.1016/J.TCS.2023.114030.

33 Sebastian Rudolph. Succinctness and tractability of closure operator representations. *Theoretical Computer Science*, 658:327–345, 2017. doi:10.1016/J.TCS.2015.12.028.

34 Luigi Santocanale and Friedrich Wehrung. Lattices of regular closed subsets of closure spaces. *International Journal of Algebra and Computation*, 24(07):969–1030, 2014. doi:10.1142/S021819671450043X.

35 Yann Strozecki. Enumeration complexity. *Bulletin of EATCS*, 1(129), 2019.

36 Shuji Tsukiyama, Mikio Ide, Hiromu Ariyoshi, and Isao Shirakawa. A new algorithm for generating all the maximal independent sets. *SIAM Journal on Computing*, 6(3):505–517, 1977. doi:10.1137/0206036.

37 Marcel Wild. A theory of finite closure spaces based on implications. *Advances in Mathematics*, 108(1):118–139, 1994.

38 Marcel Wild. Computations with finite closure systems and implications. In *International Computing and Combinatorics Conference*, pages 111–120. Springer, 1995. doi:10.1007/BFB0030825.

39 Marcel Wild. The joy of implications, aka pure Horn formulas: mainly a survey. *Theoretical Computer Science*, 658:264–292, 2017. doi:10.1016/J.TCS.2016.03.018.

40 Kaiqiang Yu, Cheng Long, Shengxin Liu, and Da Yan. Efficient algorithms for maximal k-biplex enumeration. In *Proceedings of the 2022 International Conference on Management of Data*, pages 860–873, 2022. doi:10.1145/3514221.3517847.

Structural Parameterizations of Simultaneous Planarity

Thomas Depian
TU Wien, Austria

Simon D. Fink
TU Wien, Austria

Alexander Firbas
TU Wien, Austria

Robert Ganian
TU Wien, Austria

Matthias Pfretzschner
University of Passau, Germany

Ignaz Rutter
University of Passau, Germany

Abstract

Given a set of graphs on the same vertex set, the problem Simultaneous Embedding With Fixed Edges (SEFE) asks, whether there exist planar drawings of all input graphs, such that every pair of drawings coincides on their shared subgraph. It is known that SEFE is NP-complete [32], even in the so-called sunflower case, where all pairs of input graphs have the same shared graph $G_\cap$ [57]. Fink, Pfretzschner, and Rutter [26] recently initiated the study of the parameterized complexity of SEFE in the sunflower case, mainly focusing on structural parameters of $G_\cap$. In this work, we shift the focus towards parameters of the union graph $G_\cup$ that contains the edges of all input graphs. On the positive side, we establish fixed-parameter tractability for the problem with respect to the feedback edge set number of $G_\cup$. We complement this result by showing that it, surprisingly, remains NP-complete even if $G_\cup$ has constant vertex cover number. These results settle two open questions posed by Fink et al. [26].

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases SEFE, Simultaneous Planarity, Fixed-Parameter Tractability, NP-hardness

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.25

Funding Thomas Depian, Simon D. Fink, Alexander Firbas, and Robert Ganian acknowledge support from the Vienna Science and Technology Fund (WWTF) [10.47379/ICT22029]. Simon D. Fink and Robert Ganian also acknowledge support from the Austrian Science Fund (FWF) [10.55776/Y1329 and 10.55776/COE12]. Matthias Pfretzschner and Ignaz Rutter acknowledge support from the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) [541433306].

1 Introduction

The task of computing a plane drawing of a graph – or determining that none exists – has had a fundamental and broad impact on the field of theoretical computer science. An important reason why plane drawings are desirable is that they give rise to more easily readable visualizations of graphs [50, 51]. Nevertheless, in many scenarios we are faced with the issue that there isn't a single graph we would need to visualize, but rather a (possibly small) set of planar graphs that do, however, share some of their edges and vertices. Examples where this occurs include, among others, the case of dynamic networks that change over time or can be the result of attempting to reconstruct the same underlying network using

© Thomas Depian, Simon D. Fink, Alexander Firbas, Robert Ganian, Matthias Pfretzschner, and Ignaz Rutter;
licensed under Creative Commons License CC-BY 4.0

different methods. Our input is thus a sequence of $k \geq 2$ graphs $G_1, \ldots, G_k$, for which we want to compute a corresponding sequence of planar drawings $\Gamma_1, \ldots, \Gamma_k$. It is, however, not enough to independently compute these drawings. In order to allow a user to compare the graphs by studying the sequence of drawings, it is important that the drawing is stable so that commonalities and difference can be easily observed. The problem Simultaneous Embedding with Fixed Edges models this by requiring that any vertex or edge that occurs in two of the graphs G_i, G_j is represented in the same way (i.e., by the same point resp. by the same curve) in Γ_i and Γ_j. This problem is known to be NP-complete for $k \geq 3$ [32] and despite significant research effort, the complexity of the problem is still open in the case $k = 2$.

An important special case of this problem is the *sunflower case*, where it is assumed that the intersection graph $G_i \cap G_j$ is the same for all $1 \leq i < j \leq k$. In this case, there is a single graph $G_\cap$, called the *shared graph*, that is contained in each input graph G_i, $1 \leq i \leq k$. Then, the vertices and edges of $G_\cap$ are called *shared*, whereas the vertices and edges that are contained in G_i but do not belong to $G_\cap$ are called *exclusive* and they cannot occur in any input graph G_j with $j \neq i$. We refer to instances with this property as *sunflower instances* and to the corresponding problem as *sunflower SEFE* (or S-SEFE for short). Observe that for $k = 2$, all instances of SEFE are sunflower instances. Without loss of generality, the exclusive vertices can be added to the shared graph (by adding them as isolated vertices to all other input graphs). We thus assume throughout that all our input graphs share the same vertex set V. A natural way to represent such sunflower instances of SEFE is by their *union graph*, $G_\cup$, whose vertex set is V and whose edge set is partitioned into $k + 1$ sets $E_\cap, E_1, \ldots, E_k$, where $E_\cap$ contains the shared edges and E_i, $i \in [k]$ contains the exclusive edges of G_i. It is readily seen that a sunflower instance of SEFE admits a solution if and only if its union graph $G_\cup$ admits a drawing such that for each $i \in [k]$, no pair of curves representing edges from $E_\cap \cup E_i$ crosses. An illustration of the problem is provided in Figure 1.

Jünger and Schulz [41] showed that, in this setting, the problem can be formulated purely combinatorially. Namely, it is equivalent to finding (combinatorial) planar embeddings of the input graphs $G_1, \ldots, G_k$ that induce the same planar embedding on the shared graph. Here, two planar embeddings are considered the same if they have the same rotation system and the same relative positions of their connected components. This establishes a link to the *partial drawing extension problem*, which asks whether a given drawing Γ_H (or equivalently a planar embedding) of a subgraph H of an input graph G can be extended to a planar drawing (or equivalently a planar embedding) of the complete graph G without changing Γ_H. In particular, S-SEFE is equivalent to deciding whether the shared graph $G_\cap$ admits a planar embedding that can be independently extended to each input graph G_i.

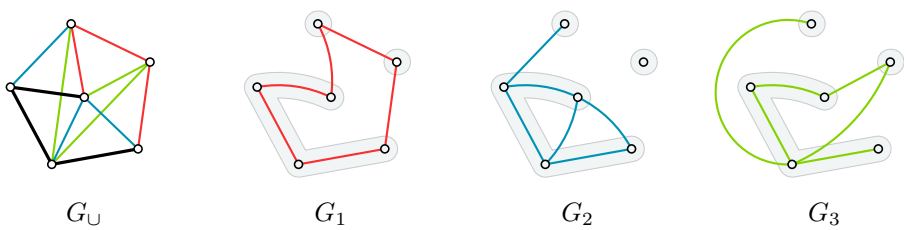


Figure 1 An instance $\mathcal{I} = (G_1, G_2, G_3)$ of S-SEFE and the corresponding union graph $G_\cup$, where the shared edges are drawn in black. Note that the drawings of G_1, G_2, and G_3 coincide on the shared graph marked in gray.

The idea of simultaneous drawings and the extension of partial drawings readily generalize other types of graph representations such as geometric intersection and contact representations. This includes, among others, (several subclasses of) interval graphs [38, 54, 55, 56, 44, 43, 45], comparability and (circular) permutation graphs [39, 42, 49], circle graphs [19, 16], and circular-arc graphs [25], which have received considerable attention in the last years. Moreover, different drawing styles have been considered such as level-planar [3, 15] or orthogonal drawings [2, 6]. Moreover, fixed-parameter algorithms for drawing extension problems, often parameterized by the number of vertices and/or edges that still need to be added, have received considerable attention in recent years [23, 31, 9, 22] albeit the techniques developed there do not seem readily transferable to SEFE.

Since its introduction in 2005 [24], the SEFE problem has been the subject of substantial research efforts over the last two decades. SEFE is known to be NP-complete even for $k = 3$ input graphs [32] and this extends to the sunflower case [57] even in the case where the shared graph is a tree [4]. The case $k = 2$ is open and there is a plethora of work that aims at solving increasingly general special cases. Angelini, Di Battista, Frati, Jelínek, Kratochvíl, Patrignani and Rutter [5] show that S-SEFE can be solved in linear time if the embedding of the shared graph is fixed. For this case, also a full Kuratowski-type characterization of the positive instances has been given [40]. While there exist attempts to generalize such characterizations [36, 18], they seem to be inherently restricted to the biconnected case. More generally, when the embedding of the shared graph is not fixed, S-SEFE can be solved in linear time if the shared graph is biconnected [1, 35], if all connected components of the shared graph are cycles [14] and even if each connected component of the shared graph is biconnected or has maximum degree 3 [12, 57]. Most recently, it has been established that SEFE admits a polynomial-time algorithm for $k = 2$ if the shared graph is connected [28, 10, 27]. We refer to the surveys [13, 53] for a more detailed overview.

In this work, we target instances of S-SEFE beyond the case of $k = 2$. A major motivation for our results is the work of Fink, Pfretzschner and Rutter [26], who initiated the study of *parameterized* algorithms capable of solving such instances. The central question underlying their work – as well as a plethora of other studies within the domain of parameterized algorithmics – is whether we can identify one or several parameters $p_1, \ldots, p_\ell$ of the input such that every instance $\mathcal{I}$ of S-SEFE can be solved in time $f(p_1+\cdots+p_\ell)\cdot|\mathcal{I}|^{\mathcal{O}(1)}$. Algorithms exhibiting such runtime behavior are called *fixed-parameter* (with respect to $p_1 + \cdots + p_\ell$)[1]. Their main results included fixed-parameter algorithms for S-SEFE parameterized (1) by k plus the *vertex cover number* (i.e., the minimum size of a vertex cover) of $G_\cup$, and (2) by k plus the *feedback edge number* (i.e., the minimum size of a feedback edge set) of $G_\cup$. In spite of these results – and a number of supplementary results targeting the structure of the shared graph – many prominent gaps remain in our understanding of the parameterized complexity of SEFE. Two questions explicitly identified by Fink, Pfretzschner and Rutter were [26, Section 5]:

(A) Can the parametric dependence on k be avoided in fixed-parameter algorithm (1)?
(B) Can the parametric dependence on k be avoided in fixed-parameter algorithm (2)?

Contributions. We expand our understanding of the complexity of S-SEFE by settling both open questions listed above. As our first result, we answer Question (A) in the negative and prove:

[1] We assume familiarity with the basic foundations of *parameterized complexity theory*, notably including the notions of *fixed-parameter tractability*, kernelization, and paraNP-hardness [21].

▶ **Theorem 1.** S-SEFE *remains* NP*-complete even when restricted to instances where* $G_\cup$ *has vertex cover number at most* 15*.*

The reduction underlying the proof of Theorem 1 is non-trivial and apart from a careful construction, it also relies on Vizing's Theorem [58] and the edge coloring machinery introduced by Misra and Gries [48]. For our second result, we turn towards Question (B) which we answer positively:

▶ **Theorem 2.** S-SEFE *is fixed-parameter tractable when parameterized by the feedback edge number of* $G_\cup$ *alone, and in particular admits a linear kernel under this parameterization.*

We note that here a *linear kernel* means that we provide a polynomial-time procedure that reduces an instance of S-SEFE to an equivalent one whose size is linear in the feedback edge number of the union graph $G_\cup$. The result strictly improves upon the previously established kernel of Fink, Pfretzschner and Rutter [26], whose size depended both on the feedback edge number and k.

In our concluding discussion, we also provide new insights into the complexity-theoretic behavior of S-SEFE (and also SEFE) under other graph parameters. Notably, we rule out parameterized algorithms w.r.t. the graph degeneracy of the union graph, and establish an equivalence between treewidth and clique-width in the sense that algorithms and lower bounds for the considered problems transfer between these two parameters.

2 Preliminaries

Let $G = (V, E)$ be a simple graph. If there exists a path between every pair of vertices of G, then G is *connected*. A *connected component* of G is a maximal induced subgraph of G that is connected. A *cut-vertex* of G is a vertex whose removal disconnects G. A *drawing* Γ of G assigns every vertex $v \in V$ to a coordinate $\Gamma(v) \in \mathbb{R}^2$ and every edge $e \in E$ to a simple arc between $\Gamma(u)$ and $\Gamma(v)$. The drawing Γ is *planar* if no two distinct arcs intersect at an interior point. A graph is planar if it admits a planar drawing. Every planar drawing Γ of G uniquely defines a *rotation system*, i.e., the clockwise circular ordering of the incident edges (and, hence, also of the adjacent vertices) for each vertex. Moreover, Γ partitions the plane into regions called *faces*. If G is disconnected, Γ additionally defines *relative positions* for the connected components of G, that is an assignment of each connected component to a unique face of every other connected component. A *planar embedding* of G is an equivalence class of planar drawings of G that induce the same rotation system and the same relative positions.

Let $\mathcal{I} = (G_1, \ldots, G_k)$ be an instance of S-SEFE. Recall that we consider the sunflower case, where all pairs of input graphs have the same shared graph $G_\cap$. A tuple $(\Gamma_1, \ldots, \Gamma_k)$ of planar drawings of $(G_1, \ldots, G_k)$ is a *simultaneous drawing* of $\mathcal{I}$ if each Γ_i induces the same drawing on $G_\cap$. Note that $\mathcal{I}$ is a positive instance of S-SEFE if and only if $(G_1, \ldots, G_k)$ admits a simultaneous drawing. We refer to a tuple $\mathcal{E}_\cup = (\mathcal{E}_1, \ldots, \mathcal{E}_k)$ of planar embeddings of $(G_1, \ldots, G_k)$ as a *simultaneous embedding* of $\mathcal{I}$ if each $\mathcal{E}_i$ induces the same embedding on $G_\cap$. Jünger and Schulz [41, Theorem 4] showed that, for $k = 2$, the existence of a simultaneous drawing is equivalent to the existence of a simultaneous embedding. This can be straightforwardly extended to $k \geq 3$ in the sunflower case by successively applying their construction to each pair of input graphs. In this paper, we thus mainly work with simultaneous embeddings instead of drawings. We define $E_\cup \coloneqq E(G_\cup)$ and $E_\cap \coloneqq E(G_\cap)$. For a (not necessarily induced) subgraph H of $G_\cup$, we define $H_\cap \coloneqq (V(H), E_\cap \cap E(H))$ and $H_i \coloneqq (V(H), (E_\cap \cup E_i) \cap E(H))$.

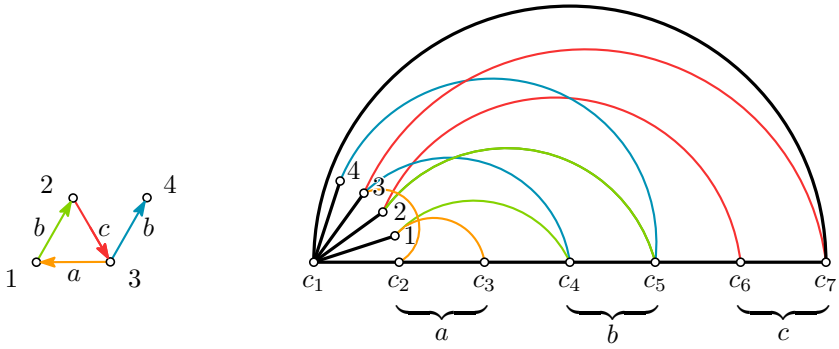


Figure 2 Example of the reduction from 2-Digraph Coloring. For simplicity, the construction is carried out with three registers a, b, c instead of 7. Left: An instance of 2-Digraph Coloring with vertices 1–4. Each color corresponds to one input graph of the S-SEFE instance. The letters a, b, c denote the register assignment for each arc. Right: The corresponding S-SEFE instance with registers a, b, c. Shared edges are drawn in black; exclusive edges are colored.

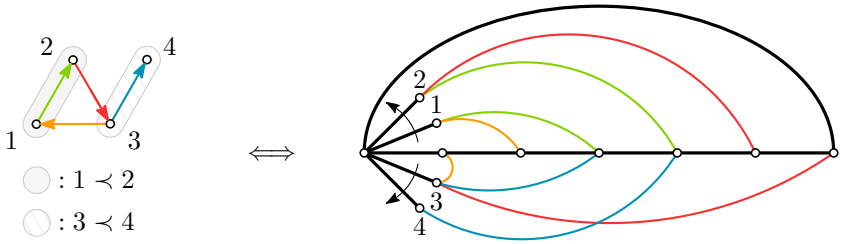


Figure 3 A solution for the 2-Digraph Coloring instance provided in Figure 2. Left: A 2-partition of the input graph's vertices and the corresponding topological orderings. Right: A simultaneous drawing of the corresponding S-SEFE instance. The black arrows indicate how to read off the 2-partition and the respective topological orderings.

For a graph G, a set $C \subseteq V$ is a *vertex cover* of G if every edge of G is incident to a vertex in C. The *vertex cover number* $\mathrm{vc}(G)$ is the size of a minimum vertex cover of G. A *feedback edge set* of G is a set $F \subseteq E$ whose removal makes G acyclic. The *feedback edge set number* $\mathrm{fen}(G)$ is the size of a minimum feedback edge set of G.

A directed graph $\vec{G}$ is acyclic precisely when its vertices admit an ordering σ such that every arc uv satisfies $\sigma(u) \prec \sigma(v)$. Such an ordering is referred to as a *topological ordering* of $\vec{G}$. The *underlying undirected graph* of a directed graph $\vec{G}$ is the simple graph obtained by removing the direction of every edge of $\vec{G}$.

3 ParaNP-hardness w.r.t. Vertex Cover of the Union Graph

In this section, we derive Theorem 1, that is, S-SEFE remains NP-complete even when restricted to instances where $vc(G_\cup)$ has vertex cover number at most 15.

We achieve this via a reduction from 2-Digraph Coloring. The problem asks whether, for a directed graph $\vec{G}$, there exists a 2-partition of its vertices $V_1 \dot\cup V_2$, such that both $\vec{G}[V_1]$ and $\vec{G}[V_2]$ are acyclic. The problem remains NP-complete even if $\vec{G}$ has maximum degree 6 [37, Theorem 1]; this fact will be central to our construction. Let $\vec{G}$ be such an instance; we compute our reduction as follows.

We aim to construct an instance of S-SEFE $\mathcal{I} = (G_1, G_2, \ldots, G_{|E(\vec{G})|})$. We assume without loss of generality that $\vec{G}$ contains no self-loops, for if $\vec{G}$ contains a self-loop, $\vec{G}$ is a trivial negative instance and we can output a trivial negative instance of S-SEFE. For each arc $uv \in \vec{G}$, $\mathcal{I}$ contains a separate input graph. To simplify the presentation, we will index the input graphs with the arcs of $\vec{G}$ rather than with the integers $[|E(\vec{G})|]$ in this section.

To build our instance, we will first describe how to construct the shared graph $G_\cap$ and then describe, for each input graph, how to extend $G_\cap$ to form the completed instance $\mathcal{I}$. (Technically, as the symbol $G_\cap$ is only well defined if the underlying instance fulfills the sunflower property, we first define an auxiliary graph $G^*_\cap$, extend this graph to obtain each input graph, and then argue that the so-obtained instance $\mathcal{I}$ fulfills the sunflower property, where in particular, $G_\cap = G^*_\cap$ (Lemma 3)). Note that in our construction, we require the edge coloring algorithm of Misra and Gries [48] specifically to enable the proof of Lemma 3.

We let $G^*_\cap$ be a cycle of length 15 with vertices $c_1, \ldots, c_{15}$ (which will form a vertex cover of size 15 of $G_\cup$), where c_1 has $|V(\vec{G})|$ pendant vertices. The vertex set of the constructed instance is $\{c_1, \ldots, c_{15}\} \cup V(\vec{G})$, where each $v \in V(\vec{G})$ identifies one pendant vertex of c_1.

The intuition for our construction is as follows: Each pendant vertex of c_1 needs to be drawn either "inside" or "outside" the shared cycle. This models the 2-partition of 2-Digraph Coloring. Moreover, for each arc ab, we define an input graph G_{ab} extending $G^*_\cap$ that forces that, if a and b are drawn both "inside" or both "outside" the cycle, in the cyclic order of vertices around c_1, going from c_2 "towards" the respective face ("inside" vs "outside"), a needs to appear before b, as otherwise this input graph would need to be drawn with a crossing. This models that the subgraphs induced by either partition can be sorted topologically, that is, they are acyclic. See Figure 2 for an example of the reduction and Figure 3 for an example of how a solution of 2-Digraph Coloring corresponds to a simultaneous embedding and vice versa.

Next, we construct the input graph G_{ab} for each $ab \in E(\vec{G})$. We first add all edges from $G^*_\cap$, and then proceed to add two additional edges as follows. We select two cycle vertices $c_{i(ab)}, c_{i(ab)+1}$ and add the edges $ac_{i(ab)}, bc_{i(ab)+1}$ to G_{ab}, where i is a map from $E(\vec{G})$ to $\{2, 4, 6, \ldots, 14\}$ which we will define later. We call such an adjacent pair of cycle vertices, selected by $i(\cdot)$, a *register*. Note that there are 7 registers in total: $(c_2, c_3), (c_4, c_5), \ldots (c_{14}, c_{15})$.

To compute $i(\cdot)$, that is, assign each arc of $\vec{G}$ a register, we compute an edge coloring (that is, an assignment of edges to colors such that adjacent edges are assigned different colors) of the undirected graph underlying $\vec{G}$, which we will call G. In particular, this means that if two arcs share an endpoint, they are assigned different registers, except if they form a 2-cycle: then, they are assigned a common register.

As $\vec{G}$ has a maximum degree of 6, so does G. By Vizing's theorem [58], G admits an edge coloring using at most 7 colors. Using the algorithm of Misra and Gries [48], we can compute such a coloring in polynomial time. We identify each of the at most 7 colors with a unique register. Formally, if the edge coloring assigns arc ab register (c_j, c_{j+1}), we set $i(ab) \coloneqq j$.

This concludes the construction of our instance $\mathcal{I}$. Before we establish the correctness of our reduction, we show:

▶ **Lemma 3.** *Instance $\mathcal{I}$ obeys the sunflower property, where in particular, $G_\cap = G^*_\cap$.*

Proof. We need to show that $G_{xy} \cap G_{z\alpha} = G^*_\cap$ for each $xy, z\alpha \in E(\vec{G})$, implying that $G_\cap$ is well-defined and equal to $G^*_\cap$. Suppose not.

By construction, as $G^*_\cap$ is a subgraph of G_{xy} and $G_{z\alpha}$, and the codomain of $i(\cdot)$ consists of even integers, this means $xc_{i(xy)} = zc_{i(z\alpha)}$ or $yc_{i(xy)+1} = \alpha c_{i(z\alpha)+1}$.

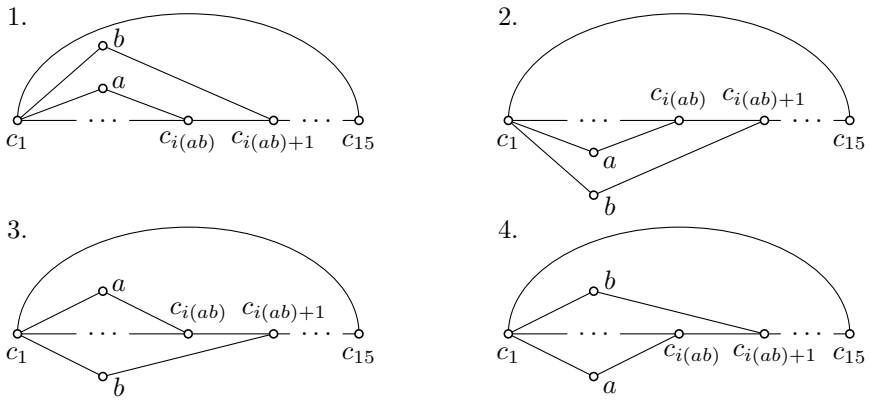


Figure 4 All 4 planar embeddings of G_{ab} for $ab \in E(\vec{G})$.

First, consider the case where $xy, z\alpha$ form a 2-cycle. Then, we have $\alpha = x$, $z = y$ and $j := i(xy) = i(z\alpha)$. In case we have $xc_{i(xy)} = zc_{i(z\alpha)}$, this means $x = y$, i.e., arc xy is a self-loop. As $\vec{G}$ does not have self-loops, this is a contradiction. Alternatively, in case we have $yc_{i(xy)+1} = \alpha c_{i(z\alpha)+1}$, this means $\alpha = z$, which again means $\vec{G}$ has a self-loop, a contradiction.

Otherwise $xy, z\alpha$ do not form a 2-cycle. First, if $xc_{i(xy)} = zc_{i(z\alpha)}$, we have $x = z$ and $i(xy) = i(z\alpha)$. But then, both y and α are neighbors of x in G. Additionally, $y \neq \alpha$, since otherwise, the two arcs would form a 2-cycle. Thus the edge coloring assigns xy and za different colors, which contradicts $i(xy) = i(z\alpha)$. The other case, $yc_{i(xy)+1} = \alpha c_{i(z\alpha)+1}$, is symmetric. ◀

To make the notion of how to "read off" a topological ordering of $c_1's$ neighbors in a simultaneous embedding precise, we define:

▶ **Definition 4.** *The* x_i-oriented, x_j-delimited *split of a cyclic permutation* $(x_1, \ldots, x_\ell)$ *where* $i \neq j$ *is given by the* left-face *total order* $x_{i-1} \prec x_{i-2} \prec \cdots \prec x_{j+1}$, *and the* right-face *total order* $x_{i+1} \prec x_{i+2} \prec \cdots \prec x_{j-1}$, *where subscripts are interpreted modulo* ℓ.

Next, we make an observation that will help us prove the correctness of our reduction. Essentially, we have that each possible embedding of G_{ab} tells us into which set of the 2-partition to assign a and b, and a guarantee that, if we assign them to the same set, the order in which we will "read off" their ordering will be consistent with there being an arc ab in the given 2-Digraph Coloring instance.

▶ **Observation 5.** *Let* $ab \in E(\vec{G})$. *The set of planar embeddings of* G_{ab} *is given by the set specified in Figure 4. Furthermore, for all planar embeddings of* G_{ab}, *we have that in both the left-face as well as the right-face total order of the* c_2*-oriented* c_{15}*-delimited split of the rotation around* c_1, *if both* a *and* b *are contained in the respective order, we have* $a \prec b$.

Now, we can show:

▶ **Lemma 6.** *The reduction is correct, that is,* $\vec{G}$ *is a positive instance of 2-Digraph Coloring if and only if* $(G_e)_{e \in E(\vec{G})}$ *is a positive instance of* S-SEFE.

Proof. ($\Rightarrow$): Let $L \dot\cup R$ be a partition of $V(\vec{G})$ such that $\vec{G}[L]$ as well as $\vec{G}[R]$ are acyclic, and let $\ell_1 \prec \cdots \prec \ell_{|L|}$ and $r_1 \prec \cdots \prec r_{|R|}$ be a topological ordering of $G[L]$ and $G[R]$, respectively, where we call the former the *left order* and the latter the *right order*.

Towards finding a simultaneous embedding of the S-SEFE instance, we first fix a planar embedding of $G_\cap$. As $G_\cap$ is connected, it suffices to specify a rotation system. Note that, as $G_\cap$ is just a cycle with pendant vertices attached to c_1, any rotation system for $G_\cap$ yields a planar embedding. For each c_i where $i \neq 1$, c_i has only two neighbors and hence, there is only one choice for c_i's rotation. For c_1, we define the following clockwise cyclic order of $N_{G_\cap}(c_1)$:

$$\left(\ell_{|L|}, \ell_{|L|-1}, \ldots, \ell_1, c_2, r_1, r_2, \ldots, r_{|R|}, c_{15}\right).$$

In other words, in the c_2-oriented c_{15}-delimited split of the rotation around c_1, the left-face order equals $\ell_1 \prec \cdots \prec \ell_{|L|}$, and the right-face order equals $r_1 \prec \cdots \prec r_{|R|}$.

It remains to find a planar embedding for each input graph G_{ab}, such that each embedding induces the same embedding on $G_\cap$. Consider G_{ab} for $ab \in E(\vec{G})$. We select a suitable embedding of G_{ab}'s four embeddings listed in Figure 4 as follows: If $a, b \in L$ and we have $a \prec b$, we use embedding 1. ($b \prec a$ is impossible since $\ell_1, \ldots, \ell_{|L|}$ is a topological ordering of $\vec{G}[L]$ and $ab \in E(\vec{G}[L])$). If $a, b \in R$ and we have $a \prec b$, we use embedding 2. ($b \prec a$ is impossible since $r_1, \ldots, r_{|R|}$ is a topological ordering of $\vec{G}[R]$ and $ab \in E(\vec{G}[R])$). If $a \in L$ and $b \in R$, we use embedding 3. Finally, if $b \in L$ and $a \in R$, we use embedding 4.

In each case, it is easy to verify that the clockwise cyclic order of c_1 in G_{ab} is a sub-order of the clockwise cyclic order of c_1 in $G_\cap$, the same holds for $c_{i(ab)}$ and $c_{i(ab)+1}$, and that the rotations for the remaining vertices coincide in the embedding of G_{ab} and the embedding of $G_\cap$. Thus, as required, we have constructed a simultaneous embedding of the S-SEFE instance.

($\Leftarrow$): Let a simultaneous embedding of the S-SEFE instance be given. Consider the c_2-oriented c_{15}-delimitted split of the clockwise order around c_1 in the planar embedding induced on $G_\cap$ by the simultaneous embedding. We denote the left-face order by $\ell_1, \ldots, \ell_{|L|}$ (with vertex set L), and denote the right-face order by $r_1, \ldots, r_{|R|}$ (with vertex set R). Observe that L and R form a partition of $V(\vec{G})$.

We claim that $\ell_1 \prec \cdots \prec \ell_{|L|}$ is a topological ordering of $\vec{G}[L]$ and that $r_1 \prec \cdots \prec r_{|R|}$ is a topological ordering of $\vec{G}[R]$, that is, $\vec{G}$ is a positive instance of 2-Digraph Coloring. For this to be the case, whenever $ab \in E(\vec{G})$ and $a, b \in L$ (resp. $a, b \in R$), we need $a \prec b$ in $\ell_1 \prec \cdots \prec \ell_{|L|}$ (resp. $r_1 \prec \cdots \prec r_{|R|}$). Let $ab \in E(\vec{G})$ and $a, b \in L$. Consider the planar embedding of G_{ab} given by the simultaneous embedding of the S-SEFE instance. By Observation 5, we have $a \prec b$. The argument for the other order is symmetric. Hence, the proof is complete. ◀

Finally, to obtain Theorem 1, we observe that the reduction can be performed in polynomial time, as the Misra–Gries edge-coloring algorithm is polynomial-time computable. Furthermore, by Lemma 3, the construction yields an S-SEFE instance satisfying the sunflower property. Moreover, the set $\{c_1, \ldots, c_{15}\}$ forms a vertex cover of $G_\cup$, so $\mathrm{vc}(G_\cup) \leq 15$. Finally, S-SEFE is known to be in NP [32]. Thus, as our reduction is correct (Lemma 6), we obtain:

▶ **Theorem 1.** S-SEFE *remains* NP*-complete even when restricted to instances where* $G_\cup$ *has vertex cover number at most* 15*.*

4 A Linear Kernel via the Feedback Edge Number of the Union Graph

In this section, we derive Theorem 2, i.e., a linear kernel for S-SEFE when parameterized by the feedback edge number of $G_\cup$. To this end, let $\mathcal{I} = (G_1, G_2, \ldots, G_k)$ be an instance of S-SEFE and let $F \subseteq E(G_\cup)$ be a minimum feedback edge set of $G_\cup$ of size $\mathrm{fen}(G_\cup)$. In the

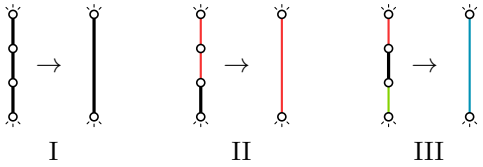


Figure 5 Illustration for Rule I, II, and Rule III of Kernel 1 used to replace ID2-paths. Shared edges are drawn in black, exclusive edges in color. Blue represents a new color not occuring in the original instance.

following, we discuss a series of reduction rules that, if applied exhaustively, transform $\mathcal{I}$ into an instance $\mathcal{I}'$ of S-SEFE , such that the size of $\mathcal{I}'$ is bounded by a function in $\text{fen}(G_\cup)$. Moreover, $\mathcal{I}'$ has a simultaneous embedding if and only if $\mathcal{I}$ does.

Before we introduce said rules, let us first discuss some useful preprocessing steps that simplify the instance $\mathcal{I}$; see also [26]. Clearly, if $G_\cup$ is disconnected, we can solve each of its connected components individually. Furthermore, Bläsius, Karrer, and Rutter showed that $\mathcal{I}$ can be decomposed at the cut-vertices of $G_\cup$ into smaller, independent sub-instances, corresponding to the split components of $G_\cup$ [11, Lemma 1]. Observe that neither of these steps increases the size of $\text{fen}(G_\cup)$. Therefore, we can assume, without loss of generality, that $G_\cup$ is biconnected. Consequently, $G_\cup - F$ is a tree with at most $4\text{fen}(G_\cup)$ vertices of degree one or at least three. However, the number of degree 2 vertices is still unbounded.

In order to also bound the number of these, we introduce the concept of an *internal-degree-2 path*, or *ID2-path* for short, which is a maximal path P in $G_\cup$ between two vertices u and v such that every internal vertex of P has degree 2 in $G_\cup$ and u and v are not adjacent in $G_\cup$. Note that this implies that P has length at least three, i.e., consists of at least two edges. Our reduction rules that we introduce in the following kernel identify such ID2-paths and carefully replace them with a single edge. These rules are centered around the insight that, from the perspective of a single graph G_i, $i \in [k]$, the path P forces its endpoints u and v to share a face in $\mathcal{E}_i$ if $E(P) \subseteq E(G_i)$. However, we can also enforce this with a single edge $uv \in E(G_i)$. Otherwise, the path P does not exist in G_i, which removes the requirement for u and v to share a face in the embedding $\mathcal{E}_i$ of G_i. Hence, we can freely choose the placement of the internal vertices of P and edges $E(P) \cap E(G_i)$ in $\mathcal{E}_i$.

▶ **Kernel 1.** Consider an instance $\mathcal{I} = (G_1, G_2, \ldots, G_k)$ of S-SEFE where $G_\cup$ is biconnected. We define the following rules, which, if applicable, map $\mathcal{I}$ to a new instance $\mathcal{I}'$ of S-SEFE. Figure 5 visualizes the rules.

Rule I: If there exists an ID2-path P in $G_\cup$ with endpoints u, v, such that $E(P) \subseteq E(G_\cap)$, then we remove all internal vertices of P from $\mathcal{I}$ and add the edge uv to every G_i, $i \in [k]$.

Rule II: If Rule I is not applicable and there exists an ID2-path P in $G_\cup$ with endpoints u, v, and some $i^* \in [k]$, such that $E(P) \subseteq E(G_{i^*})$, then we remove all internal vertices of P from $\mathcal{I}$ and add the edge uv to G_{i^*}.

Rule III: If Rules I and II are not applicable and there exists an ID2-path P in $G_\cup$ with endpoints u, v, then we remove all internal vertices of P from $\mathcal{I}$. If $E(G_i) \subseteq E(G_\cap) \cup E(P)$ for some $i \in [k]$, we also remove G_i from $\mathcal{I}$. Let $G'_\cup$ and $G'_\cap$ be the resulting union and shared graph, respectively. Afterwards, we add the graph G_P to $\mathcal{I}$ where $G_P = (V(G'_\cup), E(G'_\cap) \cup \{uv\})$.

We can readily verify that each reduction rule produces a valid instance $\mathcal{I}' = (G'_1, \ldots, G'_{k'})$ of S-SEFE with union graph $G'_\cup$ and shared graph $G'_\cap$ that obeys the sunflower property; recall that, by definition, the endpoints u and v of the ID-2 path P are not adjacent in $G_\cup$.

In Lemmas 7–9, we establish the correctness of the above-introduced reduction rules.

▶ **Lemma 7.** *Rule I of Kernel 1 is safe.*

Proof. Let $\mathcal{I}$ be an instance of S-SEFE and $\mathcal{I}'$ the instance that we obtain after applying Rule I. Furthermore, let P be the ID2-path identified when applying the rule. We now show that $\mathcal{I}$ has a simultaneous embedding if and only if $\mathcal{I}'$ does.

($\Rightarrow$): Let $(\mathcal{E}_1, \ldots, \mathcal{E}_k)$ be a simultaneous embedding of $\mathcal{I}$. As P consists solely of edges from $G_\cap$, it exists in every embedding $\mathcal{E}_i$, $i \in [k]$. We construct a simultaneous embedding of $\mathcal{I}'$ by contracting P into a single edge uv in each embedding $\mathcal{E}_i$, $i \in [k]$. This way, we ensure that all the resulting embeddings agree on $G'_\cap$. Finally, since planarity is preserved under edge contraction, we conclude that $(\mathcal{E}'_1, \ldots, \mathcal{E}'_k)$ is a simultaneous embedding of $\mathcal{I}'$.

($\Leftarrow$): Let $(\mathcal{E}'_1, \ldots, \mathcal{E}'_k)$ be a simultaneous embedding of $\mathcal{I}'$. Recall that we introduced the edge $uv \in E(G'_\cap)$ in $\mathcal{I}'$ that connects the two endpoints of P. Since $uv \in E(G'_\cap)$, the edge exists in every embedding $\mathcal{E}'_i$, $i \in [k]$. We now reintroduce P in each $\mathcal{E}'_i$ by subdividing the edge uv sufficiently many times and updating the embeddings accordingly, i.e., to reflect the subdivision process. Since the $\mathcal{E}'_i$ induce the same embedding on $G'_\cap$ before this operation, the obtained embeddings $\mathcal{E}_i$ do so on $G_\cap$ afterwards. Furthermore, subdividing an edge cannot destroy planarity, i.e., each $\mathcal{E}_i$ is a planar embedding and together they yield a simultaneous embedding of $\mathcal{I}$. ◀

▶ **Lemma 8.** *Rule II of Kernel 1 is safe.*

Proof. Let $\mathcal{I}$ be an instance of S-SEFE and $\mathcal{I}'$ the instance that we obtain after applying Rule II. Furthermore, let P denote the ID2-path and let G_{i^*} be the graph identified when applying the rule. We now show that $\mathcal{I}$ has a simultaneous embedding if and only if $\mathcal{I}'$ does.

($\Rightarrow$): Let $(\mathcal{E}_1, \ldots, \mathcal{E}_k)$ be a simultaneous embedding of $\mathcal{I}$. We now modify each embedding $\mathcal{E}_i$, $i \in [k]$ and ensure that they yield the same embedding when restricted to $G'_\cap$. For the embedding $\mathcal{E}_{i^*}$, since $E(P) \subseteq E(G_{i^*})$, we can again contract the path P into the edge uv to obtain an embedding $\mathcal{E}'_{i^*}$ of G'_{i^*}. For all other embeddings $\mathcal{E}_j$, $j \in [k] \setminus i^*$, we obtain an embedding $\mathcal{E}'_j$ of G'_j by removing the internal vertices of P. Since planarity is preserved under edge contraction and vertex deletion, each $\mathcal{E}'_i$ is a planar embedding of G'_i, $i \in [k]$. Furthermore, as we remove the same edges from $G_\cap$ from each embedding $\mathcal{E}_i$, all $\mathcal{E}'_i$, $i \in [k]$, agree on $G'_\cap$ and witness the existence of a simultaneous embedding of $\mathcal{I}'$.

($\Leftarrow$): Let $(\mathcal{E}'_1, \ldots, \mathcal{E}'_k)$ be a simultaneous embedding of $\mathcal{I}'$. We now re-insert the ID2-path P into each embedding $\mathcal{E}'_i$, $i \in [k]$. For $\mathcal{E}'_{i^*}$, we can re-insert P by subdividing the edge uv sufficiently many times and updating the embedding to reflect the subdivision. This yields a planar embedding $\mathcal{E}_{i^*}$ of G_{i^*}, which induces a (new) embedding $\mathcal{E}_\cap$ on $G_\cap$. For all other embeddings $\mathcal{E}'_j$, $j \in [k] \setminus i^*$, the fact that we could not apply Rule I tells us that there is at least one edge $e \in E(P) \setminus E(G_j)$. Hence, P is not a single path in G_j but a collection of paths and/or isolated vertices. Together with the fact that P is an ID2-path in $G_\cup$, this allows us to freely choose the embedding of the internal vertices of P and edges of $E(P) \cap E(G_\cap)$ when we insert them into $\mathcal{E}'_j$. In particular, we can choose the resulting embedding $\mathcal{E}_j$ such that its restriction to $G_\cap$ coincides with $\mathcal{E}_\cap$. Moreover, to see that $\mathcal{E}_j$ is a planar embedding, it suffices to observe that we can draw the internal vertices of P and edges of $E(G_\cap)$ arbitrarily close to u or v, respectively. We conclude that the obtained embeddings $\mathcal{E}_i$, $i \in [k]$, form a simultaneous embedding of $\mathcal{I}$. ◀

▶ **Lemma 9.** *Rule III of Kernel 1 is safe.*

Proof. Let $\mathcal{I}$ be an instance of S-SEFE and $\mathcal{I}'$ the instance that we obtain after applying Rule III. Furthermore, let P be the ID2-path identified when applying the rule. We now show that $\mathcal{I}$ has a simultaneous embedding if and only if $\mathcal{I}'$ does.

($\Rightarrow$): Let $(\mathcal{E}_1, \ldots, \mathcal{E}_k)$ be a simultaneous embedding of $\mathcal{I}$. We now modify each $\mathcal{E}_i$, $i \in [k]$, to obtain a simultaneous embedding of $\mathcal{I}'$. First, we remove the internal vertices of P from each $\mathcal{E}_i$. If $E(G_i) \subseteq E(G_\cap) \cup E(P)$, i.e., if every edge of G_i belongs to $G_\cap$ or P, we can delete the entire embedding $\mathcal{E}_i$. Since we only deleted vertices (and their incident edges), the remaining embeddings $\mathcal{E}'_i$ yield a simultaneous embedding of the corresponding sub-instance of $\mathcal{I}$. Let $\mathcal{E}'_\cap$ be the resulting embedding of $G'_\cap$. What is missing from a simultaneous embedding of $\mathcal{I}'$ is the embedding $\mathcal{E}'_P$ of the graph G_P. Recall that $E(G_P) = E(G'_\cap) \cup \{uv\}$, where u and v are the end vertices of P. We initialize $\mathcal{E}'_P = \mathcal{E}'_\cap$. This ensures that all embeddings induce the same embedding when restricted to $G'_\cap$. What is missing from an embedding of G_P is the edge uv. We now show that vertices u and v must share a face in $\mathcal{E}'_P = \mathcal{E}'_\cap$, which allows us to insert the edge uv. Towards a contradiction, assume that this would not be the case. We recall that $\mathcal{E}'_\cap$ was obtained from $\mathcal{E}_\cap$ by vertex deletion. Hence, u and v do not share a face in $\mathcal{E}_\cap$ either. However, since $(\mathcal{E}_1, \ldots, \mathcal{E}_k)$ is a simultaneous embedding of $\mathcal{I}$, and u and v are connected by the ID2-path P in $G_\cup$, there exists an edge $ab \in E(P) \cap E(G_i)$ for some $i \in [k]$ such that a and b do not share a face in $\mathcal{E}_\cap$. Since $\mathcal{E}_i$ is part of a simultaneous embedding of $\mathcal{I}$, this edge ab crosses another edge in any drawing Γ_i represented by $\mathcal{E}_i$, a contradiction to the assumption that $(\mathcal{E}_1, \ldots, \mathcal{E}_k)$ is a simultaneous embedding. Therefore, u and v share a face in $\mathcal{E}'_P$, i.e., we can insert the edge uv in $\mathcal{E}'_P$. This completes the construction of the simultaneous embedding of $\mathcal{I}'$.

($\Leftarrow$): Let $(\mathcal{E}'_1, \ldots, \mathcal{E}'_{k'})$ be a simultaneous embedding of $\mathcal{I}'$ and consider $\mathcal{E}'_P$. As $E(G_P) = E(G'_\cap) \cup \{uv\}$, the vertices u and v share a face in $\mathcal{E}'_P$. Moreover, we can identify a face f^* in $\mathcal{E}'_\cap$ which is shared by u and v and contains the edge uv (in $\mathcal{E}'_P$).

We now construct a simultaneous embedding $(\mathcal{E}_1, \ldots, \mathcal{E}_k)$ of $\mathcal{I}$ as follows. Consider an input graph G_i, $i \in [k]$, and the embedding $\mathcal{E}'_i$. For now, we assume that this embedding exists but note that our reduction rule could have removed G_i entirely from $\mathcal{I}$. This case is handled at the end. Consider the face f^* in the embedding $\mathcal{E}'_i$ when restricted to $G'_\cap$. Observe that by above arguments, the face f^* contains both u and v. Recall that we apply Rule III only if Rules I and II are not applicable. Hence, similar to the proof of Lemma 8, we observe that when induced on G_i, the ID2-path P is not a single path between u and v but a collection of paths and/or isolated vertices. Hence, we can freely choose the embedding of the respective vertices and edges when extending $\mathcal{E}'_i$ into an embedding $\mathcal{E}_i$ of G_i. In particular, we can ensure that all internal vertices of P are inside the face f^* (when $\mathcal{E}_i$ is restricted to $G_\cap$). Moreover, we can include the edges $ux, vy \in E(P)$ in the rotation around u and v, respectively, such that they lie between the same shared edges as the edge uv in $\mathcal{E}'_P$; see Figure 6. This way we ensure that the constructed embeddings will agree on $G_\cap$. Furthermore, we can draw the internal vertices of P and edges of $E(G_\cap)$ arbitrarily close to u or v, i.e., $\mathcal{E}_i$ is a planar embedding of G_i.

Finally, recall that our reduction rule can remove a G_i from $\mathcal{I}$. However, since this only happens if $E(G_i) \setminus E(G_\cap) \subseteq E(P)$, we can initialize $\mathcal{E}'_i = \mathcal{E}'_\cap$ and proceed as above. Hence, we conclude that the constructed embeddings form a simultaneous embedding of $\mathcal{I}$. ◀

With Lemmas 7–9, we have all ingredients at hand to establish fixed parameter tractability of S-SEFE when parameterized by the feedback edge number of $G_\cup$.

▶ **Theorem 2.** S-SEFE *is fixed-parameter tractable when parameterized by the feedback edge number of $G_\cup$ alone, and in particular admits a linear kernel under this parameterization.*

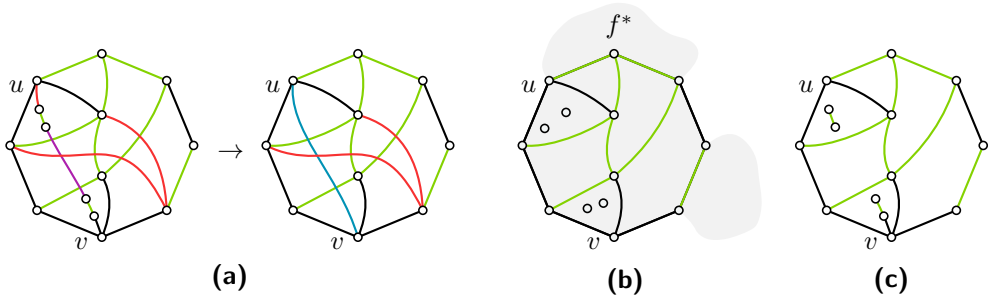


Figure 6 (a) An application of Rule III of Kernel 1, where we replace the ID2-path P from u to v with a new input graph G_P, visualized in blue. To construct the embedding $\mathcal{E}_i$, illustrated in green, the fact that $E(P) \not\subseteq E(G_i)$ allows us to freely choose the embedding of the internal vertices and respective edges of $E(P) \cap E(G_i)$. In particular, we can ensure that they **(b)** lie in the face f^* indicated in gray (when $\mathcal{E}_i$ is restricted to $G_\cap$) and that all constructed embeddings agree on $G_\cap$. **(c)** Moreover, $\mathcal{E}_i$ is a planar embedding since we can draw the inserted vertices and edges arbitrarily close to u and v, respectively.

Proof. Let $\mathcal{I} = (G_1, G_2, \ldots, G_k)$ be an instance of S-SEFE. Recall that we can assume $G_\cup$ to be biconnected thanks to known steps that simplify instances of S-SEFE [11]. Let us exhaustively apply Reduction Rules I–III of Kernel 1 to $\mathcal{I}$ to obtain $\mathcal{I}' = (G'_1, G'_2, \ldots, G'_{k'})$, $k' \geq 1$, with union graph $G'_\cup$. Note that the reduction rules maintain biconnectivity and the size of our parameter, i.e., the feedback edge number of $G_\cup$– we only replace paths where every internal vertex is of degree 2 in $G_\cup$ by a single edge.

Let $F \subseteq E(G_\cup)$ be a minimum feedback edge set of $G'_\cup$ of size $\mathrm{fen}(G'_\cup) = \mathrm{fen}(G_\cup)$. We can compute F in polynomial time by, for example, computing a spanning tree T of $G'_\cup$ and setting $F \coloneqq E(G'_\cup) \setminus E(T)$. As $G'_\cup$ is biconnected, the graph $T \coloneqq G'_\cup - F$ is a tree. To ease argumentation, we assume T to be rooted at an arbitrary leaf. Every leaf is incident to at least one edge of F – otherwise the parent of this leaf would witness the existence of a cut vertex in $G'_\cup$ which is not possible. The tree T has therefore at most $2 \cdot \mathrm{fen}(G_\cup)$ leaves and, consequently, at most $2 \cdot \mathrm{fen}(G_\cup)$ internal vertices of degree at least three. Let C denote the set of vertices $v \in V(T)$ such that v is of degree at least three in T or we have $vu \in F$ for some $u \in G'_\cup$. Observe that C contains all leaves of T and, since $|F| = \mathrm{fen}(G_\cup)$, its size is in $\mathcal{O}(\mathrm{fen}(G_\cup))$. It remains to bound the number of vertices of degree 2 in T. We do that by bounding the length of a path in T between a vertex $u \in C$ and its (first) ancestor $v \in C$ in T. Let P be such a path. We observe that P can have length at most 4. Otherwise, i.e., if P consists of 5 or more vertices, there are at least 3 consecutive internal vertices that have degree 2 in $G'_\cup$. These form a path whose endpoints are not connected in $G'_\cup$. In particular, they form an ID2-path, a contradiction to the fact that we exhaustively applied the reduction rules of Kernel 1. As there are at most $\mathcal{O}(\mathrm{fen}(G_\cup))$ such paths, one for each vertex in C (as T is a tree), each of which is of constant length, T (and thus $G_\cup$) consists of at most $\mathcal{O}(\mathrm{fen}(G_\cup))$ vertices. Moreover, as T is a tree on $\mathcal{O}(\mathrm{fen}(G_\cup))$ vertices and $\mathcal{I}$ has the sunflower property, we have for each edge $e \in E(T)$ either $e \in E(G'_\cap)$ or $e \in E(G'_i) \setminus E(G'_\cap)$ for a single $i \in [k']$. As T has $\mathcal{O}(\mathrm{fen}(G_\cup))$ edges, $|F| = \mathrm{fen}(G_\cup)$, and since we delete input graphs whose edge set coincides with that of $G'_\cap$, the number of remaining input graphs k' is in $\mathcal{O}(\mathrm{fen}(G_\cup))$. Finally, as S-SEFE is known to be in NP [32], it is decidable. With the safeness of the Rules I–III of Kernel 1 established in Lemmas 7–9, the statement follows. ◀

5 Discussion and Concluding Remarks

While Theorems 1 and 2 improve upon the previous state of the art, the parameterized complexity of S-SEFE remains wide open under other parameterizations. A key milestone left for future work is to settle whether S-SEFE is fixed-parameter tractable, XP-tractable or paraNP-hard w.r.t. k plus the *treewidth* [52] of the union graph, i.e., $\mathrm{tw}(G_\cup)$. Note that the question is only interesting if k is included in the parameterization – otherwise, we immediately obtain intractability by Theorem 1.

As a stepping stone towards understanding the problem's behavior on graphs of bounded treewidth, below we show that any treewidth-based algorithm for S-SEFE (or SEFE) would immediately also transfer to the graph parameter *clique-width*, even though on general graphs the latter can be arbitrarily smaller than the former. We remark that the formal definitions of these notions are rather technical and in fact not needed to establish the desired relationship.

▶ **Proposition 10.** *(S-)SEFE parameterized by* $\mathrm{tw}(G_\cup)+k$ *is in* FPT *(in* XP*) if and only if (S-)SEFE parameterized by* $\mathrm{cw}(G_\cup)+k$ *is in* FPT *(in* XP*, respectively).*

Proof. An FPT (or XP) algorithm parameterized by $cw(G_\cup)+k$ yields an FPT (XP, respectively) algorithm parameterized by $tw(G_\cup)+k$ due to the well-known fact that $\mathrm{cw}(G) \leq 3 \cdot 2^{\mathrm{tw}(G)-1}$ for every graph G [20]. Conversely, assume we have an FPT (or XP) algorithm for (S-)SEFE parameterized by $\mathrm{tw}(G_\cup)+k$. Consider an instance of (S-)SEFE parameterized by $cw(G_\cup)+k$. If one of the input graphs is not planar, we can reject the instance. Otherwise, k upper-bounds the thickness of $G_\cup$, i.e., the edges of $G_\cup$ can be partitioned into at most k edge sets such that each edge set induces a planar graph on $G_\cup$.

It is known that for every graph of thickness t' there exists an integer t which depends solely on the thickness of $G_\cup$ such that $G_\cup$ contains no $K_{t,t}$ as a subgraph [7]; notice here that t depends solely on k. This in turn ensures that $\mathrm{tw}(G_\cup)$ is upper-bounded by a function of $cw(G_\cup)+t$ [34], and hence also by a function of $cw(G_\cup)+k$. Therefore, the existing FPT (or XP) algorithm would imply fixed-parameter (or XP, respectively) tractability of the problem w.r.t. $\mathrm{cw}(G_\cup)+k$. ◀

A different structural parameterization that has been successfully used for some other problems is graph degeneracy [17, 46, 8]. A graph G is *d-degenerate* if every subgraph of G contains a vertex of degree at most d. The *degeneracy* of G is then the smallest d such that G is d-degenerate. Here, we provide a straightforward argument that rules out any parameterized algorithms w.r.t. graph degeneracy.

▶ **Proposition 11.** *(S-)SEFE is* NP*-hard even when restricted to instances where the union graph has degeneracy at most 2 and* $k \leq 3$*.*

Proof. We recall that SEFE and S-SEFE are known to remain NP-hard when restricted to instances satisfying $k \leq 3$ and, in particular, even $k = 3$. For each instance $\mathcal{I} = (G_1, G_2, G_3)$ of (S-)SEFE, we construct a new instance $\mathcal{I}'$ by subdividing every edge of each input graph once. Since $\mathcal{I}$ is a positive instance if and only if so is $\mathcal{I}'$ but the union graph of every constructed instance has degeneracy 2, the statement follows. ◀

Finally, we believe that a promising target for future work would be to settle the parameterized complexity of S-SEFE w.r.t. to k plus the *twin-cover number* [29, 30] (as has been asked in preceding work [26]) or k plus the *vertex integrity* of the union graph [33, 47].

References

1 Patrizio Angelini, Giuseppe Di Battista, Fabrizio Frati, Maurizio Patrignani, and Ignaz Rutter. Testing the simultaneous embeddability of two graphs whose intersection is a biconnected or a connected graph. *J. Discrete Algorithms*, 14:150–172, 2012. doi:10.1016/J.JDA.2011.12.015.

2 Patrizio Angelini, Steven Chaplick, Sabine Cornelsen, Giordano Da Lozzo, Giuseppe Di Battista, Peter Eades, Philipp Kindermann, Jan Kratochvíl, Fabian Lipp, and Ignaz Rutter. Simultaneous orthogonal planarity. In Yifan Hu and Martin Nöllenburg, editors, *Graph Drawing and Network Visualization - 24th International Symposium, GD 2016, Athens, Greece, September 19-21, 2016, Revised Selected Papers*, volume 9801 of *Lecture Notes in Computer Science*, pages 532–545. Springer, 2016. doi:10.1007/978-3-319-50106-2_41.

3 Patrizio Angelini, Giordano Da Lozzo, Giuseppe Di Battista, Fabrizio Frati, Maurizio Patrignani, and Ignaz Rutter. Beyond level planarity: Cyclic, torus, and simultaneous level planarity. *Theor. Comput. Sci.*, 804:161–170, 2020. doi:10.1016/J.TCS.2019.11.024.

4 Patrizio Angelini, Giordano Da Lozzo, and Daniel Neuwirth. Advancements on SEFE and partitioned book embedding problems. *Theor. Comput. Sci.*, 575:71–89, 2015. doi:10.1016/J.TCS.2014.11.016.

5 Patrizio Angelini, Giuseppe Di Battista, Fabrizio Frati, Vít Jelínek, Jan Kratochvíl, Maurizio Patrignani, and Ignaz Rutter. Testing planarity of partially embedded graphs. *ACM Trans. Algorithms*, 11(4):32:1–32:42, 2015. doi:10.1145/2629341.

6 Patrizio Angelini, Ignaz Rutter, and T. P. Sandhya. Extending partial orthogonal drawings. *J. Graph Algorithms Appl.*, 25(1):581–602, 2021. doi:10.7155/JGAA.00573.

7 Lowell W Beineke, Frank Harary, and John W Moon. On the thickness of the complete bipartite graph. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 60(1), pages 01–05. Cambridge University Press, 1964.

8 Matthias Bentert, Till Fluschnik, André Nichterlein, and Rolf Niedermeier. Parameterized aspects of triangle enumeration. *J. Comput. Syst. Sci.*, 103:61–77, 2019. doi:10.1016/J.JCSS.2019.02.004.

9 Sujoy Bhore, Robert Ganian, Liana Khazaliya, Fabrizio Montecchiani, and Martin Nöllenburg. Extending orthogonal planar graph drawings is fixed-parameter tractable. In Erin W. Chambers and Joachim Gudmundsson, editors, *39th International Symposium on Computational Geometry, SoCG 2023, June 12-15, 2023, Dallas, Texas, USA*, volume 258 of *LIPIcs*, pages 18:1–18:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.SOCG.2023.18.

10 Thomas Bläsius, Simon D. Fink, and Ignaz Rutter. Synchronized planarity with applications to constrained planarity problems. *ACM Trans. Algorithms*, 19(4):34:1–34:23, 2023. doi:10.1145/3607474.

11 Thomas Bläsius, Annette Karrer, and Ignaz Rutter. Simultaneous embedding: Edge orderings, relative positions, cutvertices. In Stephen K. Wismath and Alexander Wolff, editors, *Graph Drawing - 21st International Symposium, GD 2013, Bordeaux, France, September 23-25, 2013, Revised Selected Papers*, volume 8242 of *Lecture Notes in Computer Science*, pages 220–231. Springer, 2013. doi:10.1007/978-3-319-03841-4_20.

12 Thomas Bläsius, Annette Karrer, and Ignaz Rutter. Simultaneous embedding: Edge orderings, relative positions, cutvertices. *Algorithmica*, 80(4):1214–1277, 2018. doi:10.1007/S00453-017-0301-9.

13 Thomas Bläsius, Stephen G. Kobourov, and Ignaz Rutter. Simultaneous embedding of planar graphs. In Roberto Tamassia, editor, *Handbook on Graph Drawing and Visualization*, pages 349–381. Chapman and Hall/CRC, 2013.

14 Thomas Bläsius and Ignaz Rutter. Disconnectivity and relative positions in simultaneous embeddings. *Comput. Geom.*, 48(6):459–478, 2015. doi:10.1016/J.COMGEO.2015.02.002.

15 Guido Brückner and Ignaz Rutter. An spqr-tree-like embedding representation for level planarity. *J. Comput. Geom.*, 14(1):48–77, 2023. doi:10.20382/JOCG.V14I1A3.

16 Guido Brückner, Ignaz Rutter, and Peter Stumpf. Extending partial representations of circle graphs in near-linear time. *Algorithmica*, 86(7):2152–2173, 2024. doi:10.1007/S00453-024-01216-5.

17 Austin Buchanan, Jose L. Walteros, Sergiy Butenko, and Panos M. Pardalos. Solving maximum clique in sparse graphs: an $O(nm+n2^{d/4})$ algorithm for d-degenerate graphs. *Optim. Lett.*, 8(5):1611–1617, 2014. doi:10.1007/S11590-013-0698-2.

18 Johannes Carmesin and Will J. Turner. A graph minors approach to temporal sequences, 2025. arXiv:2504.00704.

19 Steven Chaplick, Radoslav Fulek, and Pavel Klavík. Extending partial representations of circle graphs. *J. Graph Theory*, 91(4):365–394, 2019. doi:10.1002/JGT.22436.

20 Derek G. Corneil and Udi Rotics. On the relationship between clique-width and treewidth. *SIAM J. Comput.*, 34(4):825–847, 2005. doi:10.1137/S0097539701385351.

21 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.

22 Thomas Depian, Simon D. Fink, Robert Ganian, and Martin Nöllenburg. The parameterized complexity of extending stack layouts. In Stefan Felsner and Karsten Klein, editors, *32nd International Symposium on Graph Drawing and Network Visualization, GD 2024, September 18-20, 2024, Vienna, Austria*, volume 320 of *LIPIcs*, pages 12:1–12:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.GD.2024.12.

23 Eduard Eiben, Robert Ganian, Thekla Hamm, Fabian Klute, and Martin Nöllenburg. Extending partial 1-planar drawings. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 43:1–43:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.ICALP.2020.43.

24 Cesim Erten and Stephen G. Kobourov. Simultaneous embedding of planar graphs with few bends. *J. Graph Algorithms Appl.*, 9(3):347–364, 2005. doi:10.7155/JGAA.00113.

25 Jirí Fiala, Ignaz Rutter, Peter Stumpf, and Peter Zeman. Extending partial representations of circular-arc graphs. In Michael A. Bekos and Michael Kaufmann, editors, *Graph-Theoretic Concepts in Computer Science - 48th International Workshop, WG 2022, Tübingen, Germany, June 22-24, 2022, Revised Selected Papers*, volume 13453 of *Lecture Notes in Computer Science*, pages 230–243. Springer, 2022. doi:10.1007/978-3-031-15914-5_17.

26 Simon D. Fink, Matthias Pfretzschner, and Ignaz Rutter. Parameterized complexity of simultaneous planarity. In Michael A. Bekos and Markus Chimani, editors, *Graph Drawing and Network Visualization - 31st International Symposium, GD 2023, Isola delle Femmine, Palermo, Italy, September 20-22, 2023, Revised Selected Papers, Part II*, volume 14466 of *Lecture Notes in Computer Science*, pages 82–96. Springer, 2023. doi:10.1007/978-3-031-49275-4_6.

27 Simon Dominik Fink and Ignaz Rutter. Constrained planarity in practice - engineering the synchronized planarity algorithm. *J. Graph Algorithms Appl.*, 29(1):91–123, 2025. doi:10.7155/JGAA.V29I1.2923.

28 Radoslav Fulek and Csaba D. Tóth. Atomic embeddability, clustered planarity, and thickenability. *J. ACM*, 69(2):13:1–13:34, 2022. doi:10.1145/3502264.

29 Robert Ganian. Twin-cover: Beyond vertex cover in parameterized algorithmics. In Dániel Marx and Peter Rossmanith, editors, *Parameterized and Exact Computation - 6th International Symposium, IPEC 2011, Saarbrücken, Germany, September 6-8, 2011. Revised Selected Papers*, volume 7112 of *Lecture Notes in Computer Science*, pages 259–271. Springer, 2011. doi:10.1007/978-3-642-28050-4_21.

30 Robert Ganian. Improving vertex cover as a graph parameter. *Discret. Math. Theor. Comput. Sci.*, 17(2):77–100, 2015. doi:10.46298/DMTCS.2136.

31 Robert Ganian, Thekla Hamm, Fabian Klute, Irene Parada, and Birgit Vogtenhuber. Crossing-optimal extension of simple drawings. 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 72:1–72:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ICALP.2021.72.

32 Elisabeth Gassner, Michael Jünger, Merijam Percan, Marcus Schaefer, and Michael Schulz. Simultaneous graph embeddings with fixed edges. In Fedor V. Fomin, editor, *Graph-Theoretic Concepts in Computer Science, 32nd International Workshop, WG 2006, Bergen, Norway, June 22-24, 2006, Revised Papers*, volume 4271 of *Lecture Notes in Computer Science*, pages 325–335. Springer, 2006. doi:10.1007/11917496_29.

33 Tatsuya Gima, Tesshu Hanaka, Masashi Kiyomi, Yasuaki Kobayashi, and Yota Otachi. Exploring the gap between treedepth and vertex cover through vertex integrity. *Theor. Comput. Sci.*, 918:60–76, 2022. doi:10.1016/J.TCS.2022.03.021.

34 Frank Gurski and Egon Wanke. The tree-width of clique-width bounded graphs without $K_{n,n}$. In Ulrik Brandes and Dorothea Wagner, editors, *Graph-Theoretic Concepts in Computer Science, 26th International Workshop, WG 2000, Konstanz, Germany, June 15-17, 2000, Proceedings*, volume 1928 of *Lecture Notes in Computer Science*, pages 196–205. Springer, 2000. doi:10.1007/3-540-40064-8_19.

35 Bernhard Haeupler, Krishnam Raju Jampani, and Anna Lubiw. Testing simultaneous planarity when the common graph is 2-connected. *J. Graph Algorithms Appl.*, 17(3):147–171, 2013. doi:10.7155/JGAA.00289.

36 Lukas Hartmann. Charakterisierung simultaner Planarität mittels verbotener Substrukturen. Master's thesis, Karlsruhe Institute of Technolgy, 2013. URL: https://i11www.iti.kit.edu/_media/teaching/theses/da-hartmann-13.pdf.

37 Ararat Harutyunyan, Michael Lampis, and Nikolaos Melissinos. Digraph coloring and distance to acyclicity. *Theory Comput. Syst.*, 68(4):986–1013, 2024. doi:10.1007/S00224-022-10103-X.

38 Krishnam Raju Jampani and Anna Lubiw. Simultaneous interval graphs. In Otfried Cheong, Kyung-Yong Chwa, and Kunsoo Park, editors, *Algorithms and Computation - 21st International Symposium, ISAAC 2010, Jeju Island, Korea, December 15-17, 2010, Proceedings, Part I*, volume 6506 of *Lecture Notes in Computer Science*, pages 206–217. Springer, 2010. doi:10.1007/978-3-642-17517-6_20.

39 Krishnam Raju Jampani and Anna Lubiw. The simultaneous representation problem for chordal, comparability and permutation graphs. *J. Graph Algorithms Appl.*, 16(2):283–315, 2012. doi:10.7155/JGAA.00259.

40 Vít Jelínek, Jan Kratochvíl, and Ignaz Rutter. A kuratowski-type theorem for planarity of partially embedded graphs. *Comput. Geom.*, 46(4):466–492, 2013. doi:10.1016/J.COMGEO.2012.07.005.

41 Michael Jünger and Michael Schulz. Intersection graphs in simultaneous embedding with fixed edges. *J. Graph Algorithms Appl.*, 13(2):205–218, 2009. doi:10.7155/JGAA.00184.

42 Pavel Klavík, Jan Kratochvíl, Tomasz Krawczyk, and Bartosz Walczak. Extending partial representations of function graphs and permutation graphs. In Leah Epstein and Paolo Ferragina, editors, *Algorithms - ESA 2012 - 20th Annual European Symposium, Ljubljana, Slovenia, September 10-12, 2012. Proceedings*, volume 7501 of *Lecture Notes in Computer Science*, pages 671–682. Springer, 2012. doi:10.1007/978-3-642-33090-2_58.

43 Pavel Klavík, Jan Kratochvíl, Yota Otachi, Ignaz Rutter, Toshiki Saitoh, Maria Saumell, and Tomás Vyskocil. Extending partial representations of proper and unit interval graphs. *Algorithmica*, 77(4):1071–1104, 2017. doi:10.1007/S00453-016-0133-Z.

44 Pavel Klavík, Jan Kratochvíl, Yota Otachi, Toshiki Saitoh, and Tomás Vyskocil. Extending partial representations of interval graphs. *Algorithmica*, 78(3):945–967, 2017. doi:10.1007/S00453-016-0186-Z.

45 Pavel Klavík and Maria Saumell. Minimal obstructions for partial representations of interval graphs. *Electron. J. Comb.*, 25(4):4, 2018. doi:10.37236/5862.

46 Michal Kotrbcík, Rastislav Královic, and Sebastian Ordyniak. Edge-editing to a dense and a sparse graph class. In Evangelos Kranakis, Gonzalo Navarro, and Edgar Chávez, editors, *LATIN 2016: Theoretical Informatics - 12th Latin American Symposium, Ensenada, Mexico, April 11-15, 2016, Proceedings*, volume 9644 of *Lecture Notes in Computer Science*, pages 562–575. Springer, 2016. doi:10.1007/978-3-662-49529-2_42.

47 Michael Lampis and Valia Mitsou. Fine-grained meta-theorems for vertex integrity. *Log. Methods Comput. Sci.*, 20(4), 2024. doi:10.46298/LMCS-20(4:18)2024.

48 Jayadev Misra and David Gries. A constructive proof of vizing's theorem. *Inf. Process. Lett.*, 41(3):131–133, 1992. doi:10.1016/0020-0190(92)90041-S.

49 Miriam Münch, Ignaz Rutter, and Peter Stumpf. Partial and simultaneous transitive orientations via modular decompositions. *Algorithmica*, 86(4):1263–1292, 2024. doi:10.1007/S00453-023-01188-Y.

50 Helen C. Purchase. Effective information visualisation: a study of graph drawing aesthetics and algorithms. *Interact. Comput.*, 13(2):147–162, 2000. doi:10.1016/S0953-5438(00)00032-1.

51 Helen C. Purchase, David A. Carrington, and Jo-Anne Allder. Empirical evaluation of aesthetics-based graph layout. *Empir. Softw. Eng.*, 7(3):233–255, 2002.

52 Neil Robertson and Paul D. Seymour. Graph minors. III. planar tree-width. *J. Comb. Theory B*, 36(1):49–64, 1984. doi:10.1016/0095-8956(84)90013-3.

53 Ignaz Rutter. Simultaneous embedding. In Seok-Hee Hong and Takeshi Tokuyama, editors, *Beyond Planar Graphs, Communications of NII Shonan Meetings*, pages 237–265. Springer, 2020. doi:10.1007/978-981-15-6533-5_13.

54 Ignaz Rutter, Darren Strash, Peter Stumpf, and Michael Vollmer. Simultaneous representation of proper and unit interval graphs. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, volume 144 of *LIPIcs*, pages 80:1–80:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.ESA.2019.80.

55 Ignaz Rutter, Darren Strash, Peter Stumpf, and Michael Vollmer. Simultaneous representation of proper and unit interval graphs. *Algorithmica*, 87(5):783–811, 2025. doi:10.1007/S00453-025-01296-X.

56 Ignaz Rutter and Peter Stumpf. Simultaneous representation of interval graphs in the sunflower case. 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 90:1–90:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ESA.2023.90.

57 Marcus Schaefer. Toward a theory of planarity: Hanani-tutte and planarity variants. In Walter Didimo and Maurizio Patrignani, editors, *Graph Drawing - 20th International Symposium, GD 2012, Redmond, WA, USA, September 19-21, 2012, Revised Selected Papers*, volume 7704 of *Lecture Notes in Computer Science*, pages 162–173. Springer, 2012. doi:10.1007/978-3-642-36763-2_15.

58 V. G. Vizing. On an estimate of the chromatic class of a p-graph. *Diskret. Analiz*, 3:25–30, 1964. in Russian.

A Tight Lower Bound for Online Service with Deadlines and Lazy Server

Yann Disser
TU Darmstadt, Germany

Linda Thelen
TU Darmstadt, Germany

Abstract

We study the online service with deadlines (or delays) problem, in which a server must serve requests for points in a metric space while balancing travel distance and promptness of service. While the problem has been extensively studied (STOC 2017), (FOCS 2019), (FOCS 2023), the main open question whether a constant competitive ratio can be achieved remains wide open. We prove a logarithmic lower bound for a natural class of algorithms already on uniform line metrics. Our lower bound applies to, and is tight for, the best known algorithms for general metrics and uniform line metrics.

2012 ACM Subject Classification Theory of computation → Online algorithms; Theory of computation → K-server algorithms

Keywords and phrases online algorithms, competitive analysis, lower bound, delay, deadlines

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.26

1 Introduction

In the *online service with delay (OSD)* problem, introduced by Azar et al. [1], requests for points of a metric space are released over time in online fashion and need to be served by a single server. The server initially occupies a point of the metric space (its *state*) and must serve each request, once it has been released, by transitioning to the requested point. Crucially, requests do not have to be served immediately, but come with individual delay cost functions that assign a waiting cost to the amount of time between release and serving of the request. The objective is to minimize the sum of the total distance traveled by the server in the metric space (*service cost*) and the waiting costs of all requests (*delay cost*). Note that the service cost does not depend on time and the delay cost does not depend on traveled distance, i.e., in particular, state transitions of the server are instantaneous.

The OSD problem formalizes in a natural way the tradeoff between batching of requests to reduce overall effort and quick response to minimize individual delay, and has sparked a series of strong results that have been very well received. In their seminal paper, Azar et al. [1] initiated the competitive analysis of OSD and presented an $\mathcal{O}(\log^4 n)$-competitive randomized algorithm for metric spaces with n points, based on a randomized embedding of the metric space into a hierarchically separated tree (HST). Later, Azar and Touitou [2] improved this result to a randomized $\mathcal{O}(\log^2 n)$-competitive algorithm, still relying on HSTs but simultaneously addressing several other problems with delay. For uniform line metrics, Bienkowski et al. [6] defined a deterministic algorithm BCKT that further improves the competitive ratio to $\mathcal{O}(\log n)$. Finally, the current state of the art was achieved in a remarkable contribution by Touitou [21], who proved a deterministic bound of $\mathcal{O}(\log n)$ for general metric spaces, by presenting a deterministic algorithm (we call it TOUITOU) that no longer relies on HSTs.

While there is a significant body of work on OSD with meaningful advances in terms of upper bounds, still no super-constant lower bounds on the competitive ratio are known. The question of whether a constant competitive ratio is possible is raised as the main open question

© Yann Disser and Linda Thelen;
licensed under Creative Commons License CC-BY 4.0

for OSD in [1, 14, 21], and the authors see evidence that a logarithmic competitive ratio might be best-possible in the fact that no better algorithms are known even for prominent special cases. For example, no better competitiveness is known even for the multilevel aggregation problem [3], a further specialization of OSD on a tree.

We consider the important special case *online service with deadlines*, where requests do not incur any waiting costs if they are served until their respective deadlines, and an unbounded cost otherwise. Note that this variant is important in its own right and no better guarantee than $\mathcal{O}(\log n)$ via Bckt [6] and Touitou [21] is known, even for uniform line metrics. We give a tight lower bound for a natural class of algorithms that, in particular, only uses uniform line metrics and applies to Bckt and Touitou.

Our Results. We consider the online service with deadlines problem on a line metric with n equidistant points. We prove a lower bound of $\Omega(\log n/\log\lambda)$ for the competitive ratio of a class of (possibly randomized) online algorithms which we call λ-lazy. These algorithms only perform actions once a request has reached its deadline, and do not serve requests that are more than λ-times farther away than this critical request. This includes every algorithm that allows only λ-times the cost of critical service (i.e., the cost of serving requests whose deadlines expired) for preemptive service to other requests. Since online service with deadlines can be seen as a special case of online service with delay, our lower bound is also valid for the online service with delay problem.

▶ **Theorem 1.** *Every $\lambda(n)$-lazy algorithm has a competitive ratio of at least $\Omega(\log n/\log\lambda(n))$ for online service with deadlines (or delays), even on a line metric with n equidistant points.*

Importantly, the notion of λ-laziness applies to the two best algorithms that are known for online service with deadlines: the Bckt algorithm by Bienkowski et al. [6] for line metrics, and the general algorithm by Touitou [21] (which we call Touitou). Since both algorithms are λ-lazy for constant λ, we can derive a logarithmic lower bound for their competitive ratios that matches the known upper bounds of $\mathcal{O}(\log n)$ for the Bckt algorithm and for the Touitou algorithm, yielding an asymptotically tight bound for the competitive ratio of these algorithms.

▶ **Corollary 2.** Bckt *and* Touitou *have a competitive ratio of $\Theta(\log n)$ for online service with deadlines (or delays) on a line metric with n equidistant points.*

Finally, we address natural generalizations of Bckt and Touitou that increase λ with growing n, by refining our notion of laziness to more accurately capture these types of algorithms. We show that the resulting class of stricly λ-lazy algorithms does not allow for better competitive ratios than $\Omega(\log n)$ either, showing that algorithms that aim to achieve a sublogarithmic competitive ratio must employ more complex strategies for preemptive service.

Related work. A natural extension of the OSD problem is to consider k-OSD with any number $k \in \mathbb{N}$ of servers. The initial result of Azar et al. [1] extends to k-OSD and yields an upper bound of $\mathcal{O}(k\log^5 n)$. For uniform metric spaces, Krnetić et al. [17] improved this to a deterministic competitive ratio of exactly $2k+1$. We can recover the classic k-server problem [18] as a special case of k-OSD by giving all requests immediate deadlines, which means that deterministic algorithms for k-OSD cannot avoid a linear dependence on k (or on $n \geq k$) in the competitive ratio [16, 18]. Note that this also applies to the algorithm of Azar et al. [1], since it only uses randomization when initially embedding the

metric in an HST. Gupta et al. [13] used randomization on weighted star metrics to obtain an $\mathcal{O}(\log k \log n)$-competitive algorithm, and Gupta et al. [14] gave an $\mathcal{O}(\operatorname{poly}\log(\Delta, n))$-competitive randomized algorithm for k-OSD with deadlines, where Δ denotes the aspect ratio between largest and smallest interpoint distance in the metric space. Note that this is compatible with weaker forms of the recently disproved randomized k-server conjecture [7], since $n \geq k$.

Another natural generalization of OSD arises when the (continuous) delay functions are revealed in online fashion, i.e., when the cost of a delay in time is only revealed once this delay has irrevocably been incurred. Azar et al. [1] showed that in this *non-clairvoyant* setting, even for weighted star metrics, the competitive ratio is at least $\Omega(\sqrt{n})$. For uniform line metrics, Bienkowski et al. [6] were able to show that their algorithm is $\mathcal{O}(\log n)$-competitive, despite not requiring clairvoyance. Recently, Touitou [22] proposed an $\mathcal{O}(\sqrt{n}\log n)$-competitive algorithm for non-clairvoyant OSD on general metric spaces.

Problems closely related to OSD arise in network design with deadlines or delay, where instead of requests to visit a point in space, connectivity requests in a graph require transmitting a subgraph. Azar and Touitou [3] introduced a deterministic framework for such problems, avoiding randomized HST embeddings, while Touitou [20] proposed one for the non-clairvoyant setting.

A prominent problem that falls in this framework is the multilevel aggregation (MLA) problem, where leaves of a rooted tree are requested and can be served in bulk by paying for a subtree that includes them and the root. This problem is a specialization of OSD on a tree, where we force the server to return to the root after each service (via urgent requests). The problem was introduced by Charikar et al. [4] with an $\mathcal{O}(D^4 2^D)$-competitive algorithm, where D is the depth of the tree, which was later improved to $\mathcal{O}(\log n)$ [2, 3, 9]. Special cases of MLA include TCP acknowledgment [10, 15] and joint replenishment [5, 8], with the underlying rooted tree being of depth 1 and 2, respectively.

Other related network design problems include node-weighted Steiner forest and directed Steiner tree with deadlines or delay, also with logarithmic upper bounds [3]. In contrast to MLA, super-constant lower bounds are known for these problems [19].

The reordering buffer management (RBM) problem is also closely related to OSD but limits the number of waiting requests instead of charging delay costs. It has been studied in general metrics by Englert et al. [11] and on line metrics by Gamzu and Segev [12], with a best known upper bound of $\mathcal{O}(\log n)$, similar to OSD.

2 Preliminaries

We consider *online service with deadlines*, where requests $r_{j\in\{1,\dots,m\}}$ for points of a metric space $(\mathcal{M}, d)$ are issued over time and need to be served by a single server. Each request r is associated with a tuple $(x_r; a_r, t_r)$ consisting of its location $x_r \in \mathcal{M}$, its arrival time $a_r \geq 0$ and its deadline $t_r \geq a_r$. To make notation more concise, we identify each request with its location.

A solution to this problem is given by a schedule $\pi = ((p_i, t_i) \in \mathcal{M} \times \mathbb{R}_{\geq 0})_{i\in\{0,\dots,N\}}$, where p_0 is the origin of the server, we demand $0 = t_0 \leq t_1 \leq \dots \leq t_N$, and the interpretation is that the server moves (instantaneously) from $p_{i-1} \in \mathcal{M}$ to $p_i \in \mathcal{M}$ at time t_i for $i \in \{1, \dots, N\}$. For each request r, we define $\pi(r) \in \mathbb{R}_{\geq 0}$ to be the earliest time that the server visits its location after time a_r, and consider r served at time $\pi(r)$; in particular, requests that arrive at the server's location are served immediately. Our goal is to minimize

$$\begin{aligned} \min \quad & \sum_{i=1}^{N} d(p_i, p_{i-1}) \\ \text{s.t.} \quad & \pi(r_j) \leq t_{r_j} \forall j \in \{1, \ldots, m\}. \end{aligned}$$

Note that the constraints can be modeled by an additive term $\sum_{j=1}^{m} c_{r_j}(\pi(r_j) - a_{r_j})$ in the objective, where the delay cost functions c_r are given by $c_r(t) = 0$ if $t \leq t_r - a_r$ and $c_r(t) = \infty$ otherwise. In that sense, the online service with deadlines problem is a specialization of the more general online service with delay problem.

For clarity, we assume that deadlines are distinct among all requests, otherwise a total order may be chosen by arbitrary tie-breaking. The events at a time t occur in the following three phases: First, all new requests arrive, then the server moves (instantaneously), serving each request it visits, and, finally, the requests' deadlines expire. We denote the position of the server at the start of time t by s_t^{start}, and the position at the end of time t by s_t^{end}. We use S_t to refer to the set of points that the server visits during time t, i.e., formally $S_t = \{p_i \mid i \in 0, \ldots, N : t_i = t\}$. The set of open (i.e., released but unserved) requests after release of new requests but before the server's movements is denoted by R_t.

Once a request has reached its deadline, we refer to it as *critical*. We say that a request is served *preemptively* if it is served strictly before its deadline.

Online service with deadlines is an online problem in the sense that requests are only revealed once they are released. We evaluate the performance of an online algorithm Alg in terms of competitive analysis, i.e., for each request sequence σ, we compare the cost $\textsc{Alg}(\sigma)$ of its solution to the cost $\textsc{Opt}(\sigma)$ of an optimum offline solution that uses knowledge of the entire input from the beginning. The *competitive ratio* ρ of Alg is then defined as $\rho := \sup_\sigma[\textsc{Alg}(\sigma)/\textsc{Opt}(\sigma)]$.

3 A logarithmic lower bound

We show a lower bound on the competitive ratio for a class of algorithms for online service with deadlines which we call λ-*lazy algorithms* (see Algorithm 1). These algorithms only move the server once some request is critical, which is without loss of generality, since delaying actions while no deadline has expired does not incur any additional cost, but reveals more information about the input sequence. If there is no other request within λ-times the distance between server and critical request, the algorithm serves only the critical request and moves towards it. Note that this definition allows for preemption, e.g. in the case where we allocate a budget of λ-times the cost of critical service for preemptive service. We say that an algorithm for online service with deadlines is $\lambda(n)$-lazy for a given function $\lambda : \mathbb{N} \to \mathbb{N}$ (possibly on a specific class of metric spaces) if it fits this description with parameter $\lambda(n)$ on metric spaces with n points. Note that we allow randomization, e.g., in choosing requests for preemptive service or the next server position. The procedure UponDeadline in Algorithm 1 is triggered at the beginning of the second phase of time t_r for each request r, i.e., after new requests are already released, but before the server has to move and deadlines expire.

▶ **Definition 3.** *We call a request $r \in R_t$* λ-isolated *at time t if every $q \in R_t \setminus \{r\}$ satisfies $d(q, s_t^{\text{start}}) \geq \lambda d(r, s_t^{\text{start}})$.*

Algorithm 1 λ-Lazy Algorithm.

Function UponDeadline(r)**:**
  if r *is* λ*-isolated***:**
    serve only r
    move server to some s_t^{end} with $d(s_t^{\text{end}}, r) = 0$ or $d(s_t^{\text{end}}, r) < d(s_t^{\text{start}}, r)$
  else
    (do something else, e.g. serve requests or move the server)

▶ Remark 4. The requirement to strictly reduce the distance to the critical request can be relaxed to allow any end position satisfying $d(s_t^{\text{end}}, r) \leq d(s_t^{\text{start}}, r)$ by repeating requests in the construction below that lure the algorithm to a specific position. For this, observe that any competitive algorithm must eventually move the server to the position of the repeated requests. To simplify presentation, we will analyze only the case where the distance to isolated requests strictly decreases.

3.1 Adversarial construction

For given $\lambda \in \mathbb{N}$, we define adversarial constructions C_λ^ℓ of a *level* $\ell \in \mathbb{N}_0$ that determines the number of points on the line: C_λ^ℓ uses $(4\lambda+4)^\ell + 1 =: w_\ell + 1$ points. Additionally, we denote by $\tilde{C}_\lambda^\ell$ a variation of C_λ^ℓ that is used to recursively construct adversarial inputs of higher levels. For the start location of the server, we choose position w_ℓ.

We will show later on that every λ-lazy algorithm serves each request at the time of its deadline, while the offline optimum solution can strategically serve some requests at an earlier time, when the server is closer to these requests.

Intuitively, both C_λ^ℓ and $\tilde{C}_\lambda^\ell$ are designed to move the algorithm from the right end of the line to the left, while incurring additional cost logarithmic in the distance between right and left end. The offline optimum solution can serve all requests of the adversarial constructions at cost linear in the number w_ℓ of points of the line segment. The construction $\tilde{C}_\lambda^\ell$ mainly consists of repeating $C_\lambda^{\ell-1}$ and $\tilde{C}_\lambda^{\ell-1}$ to create a sequence of requests that move the algorithm through a larger number of points at roughly the same competitive ratio as $C_\lambda^{\ell-1}$. The request sequence C_λ^ℓ includes additional requests compared to $\tilde{C}_\lambda^\ell$ that cause the algorithm to move the full length of the line segment again after initially reaching the left end. That way, the algorithm suffers an additional penalty linear in the number of points on the line segment, i.e., a constant is added onto the competitive ratio. As the number of points grows exponentially in the level ℓ and the competitive ratio grows linearly, the resulting competitive ratio is logarithmic in the number of points on the line.

Base case ($\ell = 0$)

We define

$$C_\lambda^0 := \tilde{C}_\lambda^0 := \{(0; 0, 0)\} .$$

The deadline and the arrival time of the request coincide, meaning that every algorithm has to serve the request upon arrival. The server's starting position is at $w_0 = 1$.

For the remainder of the construction, we define $t_\ell := |C_\lambda^\ell|$ and $\tilde{t}_\ell := |\tilde{C}_\lambda^\ell|$ as the *duration* of C_λ^ℓ and $\tilde{C}_\lambda^\ell$, respectively. The inputs C_λ^ℓ and $\tilde{C}_\lambda^\ell$ will be constructed in such a way that every request reaches its deadline in the time interval $[0, t_\ell)$ or $[0, \tilde{t}_\ell)$, respectively.

Recursive construction ($\ell > 0$)

We define C_λ^ℓ and $\tilde{C}_\lambda^\ell$ based on the constructions $\tilde{C}_\lambda^{\ell-1}$ and $C_\lambda^{\ell-1}$ of the previous level. Refer to Figures 1 and 2 for an illustration of the recursive construction. We denote by $C_\lambda^{\ell-1}+(x,t)$ (resp. $\tilde{C}_\lambda^{\ell-1}+(x,t)$) the input sequence that results from shifting all requests of $C_\lambda^{\ell-1}$ (resp. $\tilde{C}_\lambda^{\ell-1}$) by x points on the line and delaying the requests' arrival time and deadline by t time units, i.e.

$$C+(x,t) \coloneqq \{(x_r+x; a_r+t, t_r+t) \mid r \in C\} \ .$$

We define

$$\tilde{C}_\lambda^\ell \coloneqq \bigcup_{i=0}^{2\lambda+1} \left(\tilde{C}_\lambda^{\ell-1} + \left(w_\ell - (i+1)\cdot w_{\ell-1}, i\cdot \tilde{t}_{\ell-1}\right)\right) \cup \bigcup_{i=2\lambda+2}^{4\lambda+3} \left(C_\lambda^{\ell-1} + \left(w_\ell - (i+1)\cdot w_{\ell-1}, (i-2\lambda-2)\cdot t_{\ell-1} + (2\lambda+2)\tilde{t}_{\ell-1}\right)\right) \ .$$

The adversarial input C_λ^ℓ has an additional request at the starting position that reaches its deadline towards the end of the construction. We call this request the *background request* $r_{\text{bg}} = (w_\ell; 2\lambda+2, \tilde{t}_\ell)$. This request can be served early by an offline algorithm at small additional cost, while a λ-lazy algorithm will serve it only upon its deadline, incurring large additional cost. Multiple later requests then force the server back to point 0:

$$C_\lambda^\ell \coloneqq \tilde{C}_\lambda^\ell \cup \{r_{\text{bg}}\} \cup \left\{(0; \tilde{t}_\ell + i, \tilde{t}_\ell + i) \mid i = 1, \ldots, 2w_\ell\right\} \ .$$

The definition of λ-lazy algorithms only requires the server to move towards the critical request by at least one unit of distance. Therefore, it is necessary to issue multiple requests at the end of the construction (namely, as many as the maximal possible distance of the server to point 0) to ensure that the server is stationed in point 0 after the construction is finished.

3.2 Analysis

We now analyze the cost of the offline optimum solution, as well as a λ-lazy algorithm on the adversarial inputs, proving Theorem 1. We start by showing that the cost of the offline optimum solution grows linearly with the number of points on the line, while every λ-lazy algorithm has to take a detour to serve additional requests from C_λ^ℓ, thus incurring $\Omega(\ell)$ times the cost of the offline optimum solution.

▶ **Lemma 5.** *For every $\lambda \in \mathbb{N}$ and $\ell \in \mathbb{N}$, the offline optimum solution* Opt *for online service with deadlines satisfies*

$$\textsc{Opt}(C_\lambda^\ell) \leq 2w_\ell \ .$$

Proof. We recursively construct a (suboptimal) offline solution Off that achieves the claimed cost for C_λ^ℓ. We construct solutions for C_λ^ℓ and $\tilde{C}_\lambda^\ell$ for each $\ell \in \mathbb{N}$ that will be used recursively to solve constructions of higher level. Off serves each request upon arrival, and takes a preemptive detour at the start to ensure that all additional requests from C_λ^ℓ can be served at no additional cost. Note that this detour could be omitted when analyzing only $\tilde{C}_\lambda^\ell$, but we will include it to simplify analysis of the recursive constructions.

We show that Off maintains the following invariants for $\ell \in \mathbb{N}$, given that Off starts at position $s_0^{\text{start}} = w_\ell$:

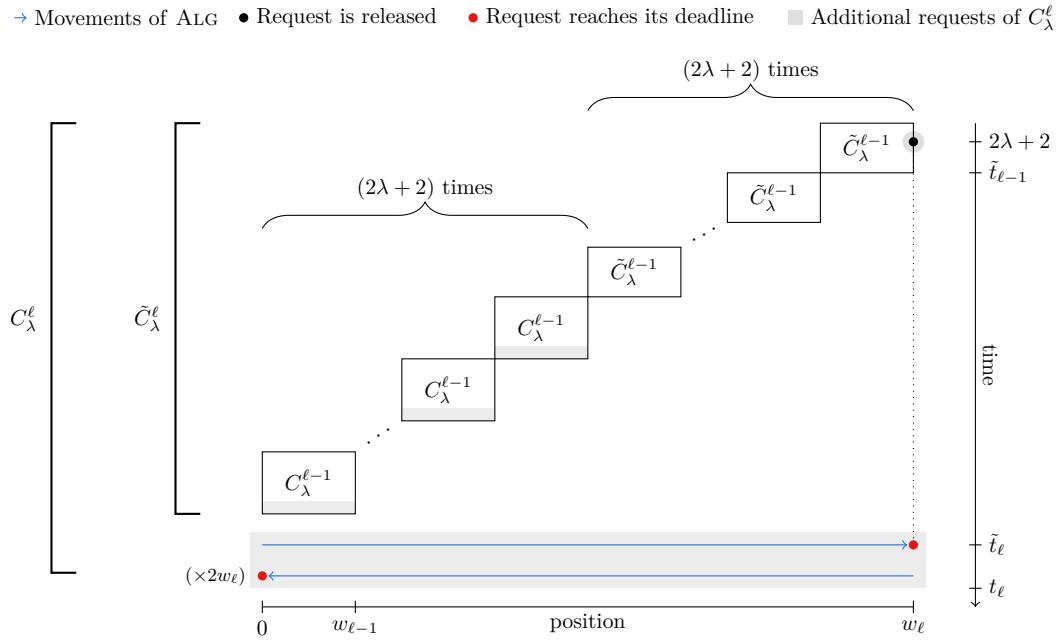


Figure 1 Recursive construction of adversarial inputs C_λ^ℓ and $\tilde{C}_\lambda^\ell$ for a λ-lazy algorithm ALG.

(I1) $\text{OFF}(C_\lambda^\ell) = \text{OFF}(\tilde{C}_\lambda^\ell) = 2w_\ell$.
(I2) OFF's server satisfies $w_\ell \in S_{2\lambda+2}$.
(I3) For each $t \in [\tilde{t}_\ell, t_\ell]$, OFF's server satisfies $s_t^{\text{start}} = s_t^{\text{end}} = 0$.

Base case ($\ell = 1$)

Refer to Figure 2 for an illustration of OFF on C_λ^1. We let OFF serve each request from copies of $\tilde{C}_\lambda^0$ upon arrival, incurring distance cost $2\lambda + 2$. At time $2\lambda + 2$, OFF returns from position $2\lambda + 2$ to $w_1 = 4\lambda + 4$, and back to $2\lambda + 2$ (recall that the server may move instantaneously), proving (I2). Note that it can therefore serve the background request at w_1 from C_λ^1 at time $2\lambda+2$ at no additional cost. OFF then continues to serve the remaining $2\lambda+2$ requests from copies of C_λ^0 upon arrival, each incurring one unit of distance cost. As required by (I1), the total cost of OFF is

$$4 \cdot (2\lambda + 2) = 2 \cdot w_1 \,.$$

OFF ends at position 0 at time $(2\lambda + 1) \cdot t_0 + (2\lambda + 2)\tilde{t}_0 = \tilde{t}_1 - 1$. It can then serve all remaining requests in 0 upon arrival at no additional cost without moving the server, hence (I3) is satisfied for C_λ^1. (I1) is maintained as before.

Inductive step ($\ell > 1$)

OFF handles each construction $C_\lambda^{\ell-1}$, $\tilde{C}_\lambda^{\ell-1}$ as in the previous recursion step, each at cost $2w_{\ell-1}$. This is possible since the required start positions align with the end position of OFF by (I3) and choice of the offsets. (I2) is maintained recursively, and the additional requests in C_λ^ℓ are served at no additional cost, since OFF is at their location at the time of their arrival by (I2) and (I3). OFF may therefore remain at position 0 after arriving there before time $\tilde{t}_\ell$, proving (I3). As required by (I1), the total cost is

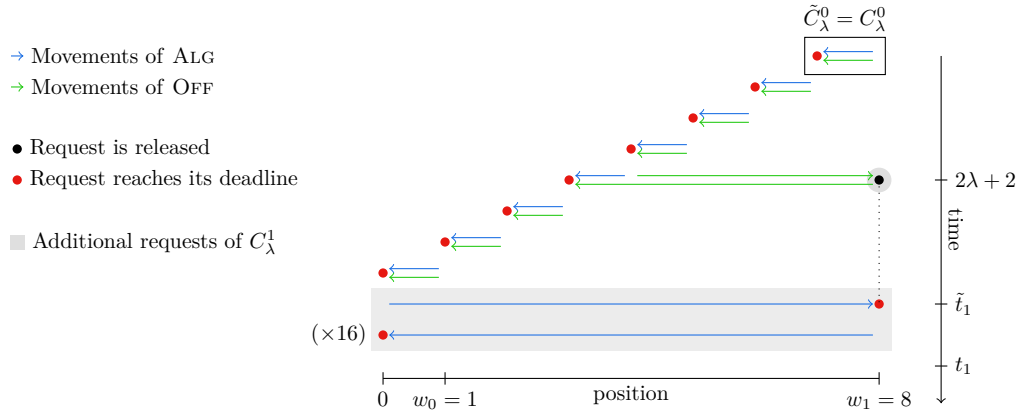


Figure 2 Illustration of C^1_λ for $\lambda = 1$, a λ-lazy algorithm ALG and the offline solution OFF.

$$\begin{aligned}\textsc{Off}(C^\ell_\lambda) = \textsc{Off}(\tilde{C}^\ell_\lambda) &= (2\lambda+2)\textsc{Off}(\tilde{C}^{\ell-1}_\lambda) + (2\lambda+2)\textsc{Off}(C^{\ell-1}_\lambda) \\ &\overset{\text{(I1)}}{=} (4\lambda+4)2w_{\ell-1} = 2w_\ell\,.\end{aligned}$$

◀

▶ **Lemma 6.** *For every $\lambda \in \mathbb{N}$ and $\ell \in \mathbb{N}_0$, every λ-lazy algorithm* ALG *satisfies*

$$\textsc{Alg}(\tilde{C}^\ell_\lambda) \geq \frac{\ell}{2} w_\ell \quad \textit{and} \quad \textsc{Alg}(C^\ell_\lambda) \geq \left(\frac{\ell}{2}+1\right) w_\ell\,.$$

Proof strategy and intuition. We prove the claim by induction on ℓ. In particular, we show that we can analyze copies of $C^{\ell-1}_\lambda$ and $\tilde{C}^{\ell-1}_\lambda$ independently in the recursive construction. To apply this argument, we consider the more general setting where C^ℓ_λ or $\tilde{C}^\ell_\lambda$ are used as part of a larger construction (i.e., there may exist other requests) and start at some later start time t_{start}. Note also that all arguments are translation invariant and can therefore also be applied to shifted copies $C^\ell_\lambda + (x,t)$ and $\tilde{C}^\ell_\lambda + (x,t)$. We will see that ALG serves exactly the requests from C^ℓ_λ (resp. $\tilde{C}^\ell_\lambda$) in the time interval $[t_{\text{start}}, t_{\text{start}} + t_\ell)$ (resp. $[t_{\text{start}}, t_{\text{start}} + \tilde{t}_\ell)$). Since copies of C^ℓ_λ and $\tilde{C}^\ell_\lambda$ are used t_ℓ or $\tilde{t}_\ell$ time units apart, it is then easy to see that ALG handles each copy independently.

To apply the induction hypothesis, we impose a set of assumptions which ensure that requests outside of C^ℓ_λ and $\tilde{C}^\ell_\lambda$ do not interfere with our intended behavior of ALG on C^ℓ_λ and $\tilde{C}^\ell_\lambda$. Intuitively, we assume that no outside request becomes critical during the execution of C^ℓ_λ or $\tilde{C}^\ell_\lambda$ (which would trigger an unexpected service), and that no outside requests are too close by, so that the requests from C^ℓ_λ and $\tilde{C}^\ell_\lambda$ are isolated when becoming critical. Additionally, we require that ALG's server is at the correct position at the start of the execution of C^ℓ_λ or $\tilde{C}^\ell_\lambda$. More formally, we make the following assumptions:

(AStartPosition) ALG's server is at position w_ℓ at the start of time t_{start}.

(ANoDelay) During time $[t_{\text{start}}, t_{\text{start}} + t_\ell)$ (resp. $[t_{\text{start}}, t_{\text{start}} + \tilde{t}_\ell)$), no requests other than those from C^ℓ_λ (resp. $\tilde{C}^\ell_\lambda$) reach their deadlines.

(AIsolationArea) It holds that

$$\begin{aligned} R_t \cap (-\infty, w_\ell + 2\lambda + 2) &\subseteq \tilde{C}^\ell_\lambda \quad && \forall t \in [t_{\text{start}}, t_{\text{start}} + \min\{\tilde{t}_\ell, 2\lambda+2\}) \\ R_t \cap (-\infty, w_\ell) &\subseteq \tilde{C}^\ell_\lambda && \forall t \in [t_{\text{start}} + 2\lambda + 2, t_{\text{start}} + \tilde{t}_\ell)\,, \end{aligned}$$

and, for C^ℓ_λ,

$$R_t \cap (-\infty, (2\lambda+2)w_\ell) \subseteq C^\ell_\lambda \qquad \forall t \in [t_{\text{start}} + \tilde{t}_\ell, t_{\text{start}} + t_\ell)\,.$$

Intuitively, (AIsolationArea) ensures that requests outside those from $\tilde{C}_\lambda^\ell$ or C_λ^ℓ are far enough to not interfere with the behavior of a λ-lazy server on the requests from the constructions. More precisely, we require that during the first $2\lambda+2$ time units of the constructions, no outside requests are less than $2\lambda+2$ points to the right of the starting point, in order not to interfere with the first requests of $\tilde{C}_\lambda^\ell$. After that, when the server has already moved sufficiently far from the starting point, we weaken this requirement and demand only that no outside requests lie to the left of the starting point. For C_λ^ℓ, since the background request forces the server to move back to the starting point later on, we require an isolation area of $(2\lambda+2)$-times the region that requests become critical in during the shifted time interval $[\tilde{t}_\ell, t_\ell)$, to account for the unknown position of the server.

Under these assumptions, we show that the following invariants are maintained:

(ICost) ALG incurs distance cost of at least $\ell/2 \cdot w_\ell$ during time $[t_{\text{start}}, t_{\text{start}} + \tilde{t}_\ell)$ and $(\ell/2+1)w_\ell$ during time $[t_{\text{start}}, t_{\text{start}} + t_\ell)$ (to serve requests from $\tilde{C}_\lambda^\ell$ and C_λ^ℓ, respectively).

(IIsolation) All requests from C_λ^ℓ and $\tilde{C}_\lambda^\ell$ are λ-isolated and critical when ALG serves them.

(IEndPosition) ALG's server is at position 0 at the end of time $t_{\text{start}} + \tilde{t}_\ell - 1$, and at the end of time $t_{\text{start}} + t_\ell - 1$ for C_λ^ℓ.

For the proof, we set $t_{\text{start}} = 0$ for simplicity of notation, but note that other requests may have been released before time t_{start}.

Proof.

Base case ($\ell = 0$)

We first consider the base constructions C_λ^0 and $\tilde{C}_\lambda^0$, which both consist only of a single request $r = (0; 0, 0)$. By (AStartPosition), the algorithm's server starts at position $s_0^{\text{start}} = w_0 = 1$.

The request r is λ-isolated at time $t = 0$, since, by (AIsolationArea), there is no other open request within $[1-\lambda, 1+\lambda]$. As r also reaches its deadline at time 0, r is served, proving (IIsolation). Since $s_t^{\text{start}} = 1$, we have $d(r, s_t^{\text{start}}) = 1$, and λ-lazy algorithms have to reduce the distance to isolated critical requests if possible. Therefore, the only valid next server position is position $0 = s_t^{\text{end}}$, proving (IEndPosition) (recall that $t_0 = \tilde{t}_0 = 1$). To serve r, the server had to incur one unit of distance cost, proving (ICost).

Inductive step ($\ell \geq 1$)

We will prove the induction invariants on C_λ^ℓ and $\tilde{C}_\lambda^\ell$ by applying the induction hypothesis to the copies of $C_\lambda^{\ell-1}$ and $\tilde{C}_\lambda^{\ell-1}$. To this end, we first verify that the assumptions on the copies of $C_\lambda^{\ell-1}$ and $\tilde{C}_\lambda^{\ell-1}$ hold, and then show that, using the induction hypothesis, the invariants remain valid for $\tilde{C}_\lambda^\ell$ and C_λ^ℓ.

Assumptions on $\tilde{C}_\lambda^{\ell-1}$. We constructed C_λ^ℓ and $\tilde{C}_\lambda^\ell$ in such a way that all requests reach their deadlines within t_ℓ (resp. $\tilde{t}_\ell$) time units after the start of the construction. Since all copies of $C_\lambda^{\ell-1}$ (resp. $\tilde{C}_\lambda^{\ell-1}$) used in $\tilde{C}_\lambda^\ell$ are $t_{\ell-1}$ (resp. $\tilde{t}_{\ell-1}$) time units apart, no requests from previous copies remain open at the start of any copy of a construction of level $\ell-1$. Moreover, since the background request r_{bg} from C_λ^ℓ is released only at time $2\lambda+2$, it does not interfere with (AIsolationArea). It follows that (ANoDelay) and (AIsolationArea) carry through from $\tilde{C}_\lambda^\ell$ or C_λ^ℓ to the copies of $\tilde{C}_\lambda^{\ell-1}$. Furthermore, (AStartPosition) on $\tilde{C}_\lambda^\ell$ or C_λ^ℓ implies (AStartPosition) for the first copy of $\tilde{C}_\lambda^{\ell-1}$. It follows that we can apply the induction hypothesis to the first copy of $\tilde{C}_\lambda^{\ell-1}$. Specifically, (IEndPosition) then implies (AStartPosition) for the next copy of $\tilde{C}_\lambda^{\ell-1}$. By repeated application of the invariants, (AStartPosition) is maintained for each copy of $\tilde{C}_\lambda^{\ell-1}$, and we can apply the induction hypothesis.

Assumptions on $C_\lambda^{\ell-1}$. For the copies of $C_\lambda^{\ell-1}$, observe that (ANoDelay) and (AStartPosition) are maintained by the same reasoning. For (AIsolationArea), note that it is sufficient to show that there are no requests $r \le (2\lambda+2)w_{\ell-1}+w_\ell-(2\lambda+3)w_{\ell-1} = w_\ell - w_{\ell-1}$ aside those from copies of $C_\lambda^{\ell-1}$ during time $[(2\lambda+2)\tilde{t}_{\ell-1}, (2\lambda+2)\tilde{t}_{\ell-1}+(2\lambda+1)t_{\ell-1}) \subseteq [0,\tilde{t}_\ell)$. This is guaranteed by (AIsolationArea) on $\tilde{C}_\lambda^\ell$ or C_λ^ℓ and definition of C_λ^ℓ. Hence, the induction hypothesis can again be applied to all copies of $C_\lambda^{\ell-1}$.

Induction invariants for $\tilde{C}_\lambda^\ell$. It is easy to see that (IIsolation) carries through from $\tilde{C}_\lambda^{\ell-1}$ and $C_\lambda^{\ell-1}$ to $\tilde{C}_\lambda^\ell$, as $\tilde{C}_\lambda^\ell$ contains no other requests. For (ICost), we find that by (ICost) on the copies of $\tilde{C}_\lambda^{\ell-1}$ and $C_\lambda^{\ell-1}$ together with $w_\ell = (4\lambda+4)w_{\ell-1}$, the cost incurred by ALG to serve requests from $\tilde{C}_\lambda^\ell$ is at least

$$\begin{aligned}(2\lambda+2)\textsc{Alg}(\tilde{C}_\lambda^{\ell-1}) + (2\lambda+2)\textsc{Alg}(C_\lambda^{\ell-1}) &\overset{\text{(ICost)}}{\ge} (4\lambda+4)\frac{\ell-1}{2}w_{\ell-1} + (2\lambda+2)w_{\ell-1}\\ &= \frac{\ell}{2}w_\ell\,,\end{aligned}$$

as required.

Finally, ALG is at position $w_\ell - (4\lambda+4)w_{\ell-1} = 0$ at the end of time $(2\lambda+2)t_{\ell-1} + (2\lambda+2)\tilde{t}_{\ell-1} - 1 = \tilde{t}_\ell - 1$ by (IEndPosition) on the last copy of $C_\lambda^{\ell-1}$, showing (IEndPosition) for $\tilde{C}_\lambda^\ell$.

Induction invariants for C_λ^ℓ. First note that we can apply the induction hypothesis to $\tilde{C}_\lambda^\ell$, since all assumptions on $\tilde{C}_\lambda^\ell$ hold by definition of the additional requests from C_λ^ℓ and the assumptions on C_λ^ℓ.

By (IEndPosition) on $\tilde{C}_\lambda^\ell$, ALG is at position 0 at the end of time $\tilde{t}_\ell - 1$. As ALG is λ-lazy, the server remains there until another request reaches its deadline. The next such request is the request $r_{\text{bg}} = (w_\ell; 2\lambda+2, \tilde{t}_\ell)$. The request is not served before it reaches its deadline at time $\tilde{t}_\ell$, since by (IIsolation) on $\tilde{C}_\lambda^\ell$, only the triggering request is served in each service prior to r_{bg} becoming critical.

By (AIsolationArea), there is no request at points within $(-\infty, (2\lambda+2)w_\ell)$ at time $\tilde{t}_\ell$. Therefore, r_{bg} is λ-isolated when it becomes critical at time $\tilde{t}_\ell$. ALG then serves r_{bg}, generating at least cost w_ℓ. It follows that ALG incurs distance cost of at least $(\ell/2+1)w_\ell$ for requests from C_λ^ℓ ((ICost) is maintained).

Afterwards, ALG's server is at a position $s_{\tilde{t}_\ell}^{\text{end}} \in (0, 2w_\ell)$, since it must move towards r_{bg}. Since by (AIsolationArea), there are no open requests within $(-\infty, (2\lambda+2)w_\ell)$, each following request $q_i \coloneqq (0; \tilde{t}_\ell + i, \tilde{t}_\ell + i)$ for $i \in \{1, \dots, 2w_\ell\}$ is λ-isolated at the time it reaches its deadline. Hence, the distance of ALG's server to 0 must decrease with each served request (if possible). It follows that ALG is at position 0 after it serves q_{2w_ℓ} at time $\tilde{t}_\ell + 2w_\ell = t_\ell - 1$, proving (IEndPosition). ◀

Proof of Theorem 1. Consider any $\lambda(n)$-lazy algorithm ALG. For a line with n points, we use an adversarial construction $C_{\lambda(n)}^\ell$. Since $C_{\lambda(n)}^\ell$ needs the underlying line metric to have at least $w_\ell + 1 = (4\lambda(n)+4)^\ell + 1$ points, we pick $\ell \coloneqq \left\lfloor \log_{4\lambda(n)+4}(n-1) \right\rfloor$. Lemmas 5 and 6 now imply

$$\frac{\textsc{Alg}(C_{\lambda(n)}^\ell)}{\textsc{Opt}(C_{\lambda(n)}^\ell)} > \frac{\ell}{4} = \Omega(\log n / \log \lambda(n))\,.$$ ◀

4 Applications

In this section, we establish that the algorithms Bckt [6] and Touitou [21] are λ-lazy with constant λ on line metrics. This implies a lower bound of $\Omega(\log n)$ on the competitive ratio of the algorithms by Theorem 1. Since both algorithms have been shown have competitive ratio $\mathcal{O}(\log n)$, the lower bound is tight. The proof details are deferred to appendices A and B.

4.1 The Bckt Algorithm

The algorithm Bckt was presented by Bienkowski et al. [6] (see Algorithm 2). It is $\mathcal{O}(\log n)$-competitive for online service with delay on a line with n equidistant points. Since online service with deadlines can be seen as a special case of online service with delay, Bckt can be applied to the deadlines case by modelling deadlines as the request accumulating very large delay only at the time of its deadline. The algorithm works in phases consisting of a *waiting subphase* and a *serving subphase*. During the waiting subphase, the server waits at its current position until there is a group of requests whose summed delay cost becomes roughly equal to the cost of serving them. At this point, a serving subphase starts, during which the server serves these requests and performs preemptive service to other requests that are similarly close. Figure 3 shows an illustration of a phase of Bckt. Note that we consider the line always centered at the server's position (in terms of the numbering of its points).

To be precise, during the execution, the algorithm partitions the line into $\mathcal{O}(\log n)$ *buckets* of geometrically increasing sizes, starting at the current position of the server. The i-th bucket to the right (resp. left) of the server is the interval $[2^{i-1}, 2^i - 1]$ (resp. $[-2^i + 1, -2^{i-1}]$ on the left), assuming that the points are numbered centered at 0 at the position of the server. We denote this bucket by B_{+i} (resp. B_{-i}) and call i the *label* of $B_{\pm i}$. For a bucket B, $|B|$ denotes the number of points it contains, i.e., $|B_{+i}| = |B_{-i}| = 2^{i-1}$, and we write $w(B)$ for the total delay cost of all open requests in B, i.e., for a given time t,

$$w(B) \coloneqq \sum_{r \in R_t \cap B} \mathrm{delay}_r(t)\,,$$

where delay_r is the function mapping a time t to the accumulated delay cost of request r at time t.

A waiting subphase ends once a bucket becomes *full*, i.e., $w(B) \geq |B|$. At this point, Bckt identifies the largest quarter-full bucket, i.e., the bucket B_i with the largest label $|i|$ such that $w(B_i) \geq |B_i|/4$. We call $|i|$ the *phase label*, and B_i the *critical bucket* of the current phase. In the distinct deadlines setting, a request immediately fills up its bucket when it reaches its deadline and all other buckets are empty, hence the critical bucket is the one containing the request that has just reached its deadline.

In the following serving phase, Bckt defines the *cleaning area* as the region $\bigcup_{j=1}^{|i|+1}(B_{-j} \cup B_{+j}) = [-(2^{|i|+1} - 1), 2^{|i|+1} - 1]$. Bckt serves all requests in the cleaning area and chooses its new position to be the point $\pm 2^{|i|-1}$ within the critical bucket.

▶ **Proposition 9.** Bckt *is* 4*-lazy.*

4.2 The Touitou Algorithm

Similarly to other algorithms for online service with deadlines or delays, Touitou is divided into *services*. Every request maintains a level that corresponds to the reluctance of the algorithm to serve the request. When a request reaches its deadline, a service starts during

Algorithm 2 BCKT (adapted from [6, Algorithm 1]).

Waiting phase:

split the line into buckets $B_{\pm i} = \pm[2^{i-1}, 2^i - 1]$.
wait until there exists a bucket B with $w(B) \geq |B|$.

Serving phase:

$i \leftarrow \arg\max\{|j| \colon w(B_j) \geq |B_j|/4\}$ (B_i is the largest quarter-full bucket)
serve the area $[-2^{|i|+1} + 1, 2^{|i|+1} - 1]$.
if $i > 0$**:** move to $2^{|i|-1}$
else move to $-2^{|i|-1}$

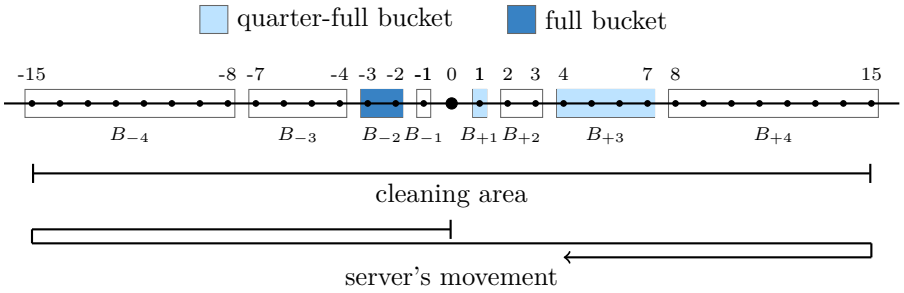


Figure 3 Illustration of the algorithm BCKT for a single phase (adapted from [6, Fig. 1.1]). Individual requests are not depicted.

which the algorithm identifies a set of requests that are eligible for service. A subset of these requests is chosen in order of increasing deadlines. Refer to Algorithm 3 for the pseudocode of TOUITOU.

To be precise, each request r maintains a level l_r that is initially set to $-\infty$. The level of a request increases only when the algorithm considers serving it, but decides not to, which is described below. If s denotes the current position of the server, the *adjusted level* of r is defined as $\bar{l}_r := \max\{l_r, \lceil \log_2 d(r, s) \rceil\}$.

When a request r reaches its deadline, a service S starts with level $l_S := \bar{l}_r + 3$. All requests with adjusted level at most l_S are eligible for service. Eligible requests are added to a list of requests for service in order of increasing deadlines. With each added request, TOUITOU computes a 2-approximation for a Steiner tree that connects the requests in the list to the current server position. Once the solution cost exceeds $4 \cdot 2^{l_S}$, the algorithm stops adding requests to the list. The requests in the list are then served on a DFS tour. Eligible requests that are not served are upgraded to level $l_S + 1$.

A service S is called *primary* if $\bar{l}_r \neq l_r$, i.e., the distance between r and the server dominates the level of r. For primary service, the server moves to r after the service. Otherwise, the server returns to its original position. Intuitively, this is the case if the request's location has been inconvenient compared to other requests in past services.

▶ **Proposition 12.** TOUITOU *is* 16*-lazy for inputs on a line metric.*

Corollary 2 follows immediately from Propositions 9 and 12 and Theorem 1.

▶ Remark 7. For the TOUITOU algorithm, the upper bound on the competitive ratio was also expressed in terms of the number m of requests as $\mathcal{O}(\log m)$. Our construction can be adapted to show that this bound is also tight. Observe that the TOUITOU algorithm

Algorithm 3 TOUITOU (adapted from [21, Algorithm 1]).

Function UponRequest(r)**:**
  $l_r \leftarrow -\infty$
Function UponDeadline(r)**:**
  start a new service S with $l_S := \bar{l}_r + 3$
  $E_S \leftarrow \{q \in R_t \mid \bar{l}_q \leq l_S\}$ (eligible requests)
  $Q_S \leftarrow \emptyset$ (requests chosen for service)
  for $q \in E_S$ *in order of increasing deadlines***:**
    add q to Q_S
    $P \leftarrow$ STEINERTREE(Q_S)
    if $\text{cost}(P) \geq 4 \cdot 2^{l_S}$**:**
      break from the loop
  serve requests in Q_S using a DFS tour on P
  for $q \in E_S \setminus Q_S$**:**
    $l_q \leftarrow l_S + 1$
  if $l_r \neq \bar{l}_r$**:** move the server to r

not only moves closer to critical requests after primary service, but moves exactly to the location of the critical request. We can therefore adapt the adversarial input so that only one request in point 0 is issued at the end of the construction to force the server to its final destination, instead of $2w_\ell$ consecutive requests. In doing so, the total number of requests in the adversarial input is reduced to $m = \mathcal{O}(n)$, implying a tight lower bound of $\Omega(\log m)$ for the competitive ratio of TOUITOU in the number of requests.

4.3 Generalizations

A natural idea to avoid the logarithmic lower bound is to adapt existing algorithms to be λ-lazy with growing values of λ. For both BCKT and TOUITOU, the parameter λ stems from a budget factor of $\Theta(\lambda)$ for preemptive service. To be precise, both algorithms serve all requests within range $c \cdot \lambda$ (for some constant $c \leq 1$) of the critical request, and serve requests within range at most λ of the critical request. We can generalize this to growing functions $\lambda(n)$ by replacing the fixed constant λ with a value depending on the size of the metric space. We can adapt Algorithm 1 accordingly to obtain a new class of *strictly* $\lambda(n)$*-lazy* algorithms (see Algorithm 4). Note that the requirement to serve (all) requests within range $c \cdot \lambda(n)$ ensures $\lambda(n)$-laziness without laziness for asymptotically smaller values of λ – otherwise we cannot hope to improve competitiveness by Theorem 1.

Algorithm 4 Strictly $\lambda(n)$-lazy Algorithm ($c \leq 1$).

Function UponDeadline(r)**:**
  serve all requests q with $d(s_t^{\text{start}}, q) \leq c \cdot \lambda(n) \cdot d(s_t^{\text{start}}, r)$
  (optionally) serve some requests q with $d(s_t^{\text{start}}, q) \leq \lambda(n) \cdot d(s_t^{\text{start}}, r)$
  move server to some s_t^{end} with $d(s_t^{\text{end}}, r) = 0$ or $d(s_t^{\text{end}}, r) < d(s_t^{\text{start}}, r)$

It is now easy to adapt the adversarial construction from Section 3 to obtain a generalized lower bound on the competitive ratio. Note that the following bound becomes $\omega(\log n)$ for $\lambda(n) \in \omega(1)$, i.e., there is nothing to be gained by scaling λ with n.

▶ Remark 8. Every strictly $\lambda(n)$-lazy algorithm has competitive ratio $\Omega(\lambda(n) \cdot \log n / \log \lambda(n))$.

Proof sketch. We can adapt the adversarial construction from Section 3 to exploit the increased budget for preemptive service. To do so, it suffices to add "decoy" requests with a late deadline within range $c \cdot \lambda(n)$ around each critical request. In addition to the factor of $\Omega(\log n / \log \lambda(n))$ proven in Theorem 1, the strictly $\lambda(n)$-lazy algorithm suffers a loss of factor $c \cdot \lambda(n)$ for preemptive service, which was not considered in the proof of Theorem 1. This results in a lower bound of $\Omega(\lambda(n) \cdot \log n / \log \lambda(n))$ on the competitive ratio of every strictly $\lambda(n)$-lazy algorithm. ◀

5 Conclusion

In this paper, we established a logarithmic lower bound on the competitive ratio of $\lambda(n)$-lazy algorithms for online service with deadlines (or delay) on uniform line metrics. This bound applies to the Bckt and Touitou algorithms, showing that their competitive ratios fall in $\Theta(\log n)$. Moreover, we demonstrated that the competitiveness of Bckt and Touitou cannot be improved by scaling the parameter λ with the size n of the metric space.

Our results highlight the inherent limitations of current algorithms and suggest that achieving a constant competitive ratio may be challenging. As evidenced in Section 4.3, avoiding the logarithmic lower bound requires an approach significantly different from the current state of the art for deterministic (or randomized) algorithms.

It remains an open question whether the super-constant lower bound demonstrated in this paper can be extended to a general bound for all online algorithms. While it may be possible to improve the upper bound of $\mathcal{O}(\log n)$, there is evidence that the bound is tight: No better deterministic or randomized algorithms are known for metric spaces other than HSTs of bounded depth [1]; the same holds for related problems, such as RBM [11, 12] and special cases of OSD such as MLA [3]. Notably, for the directed online Steiner tree problem, the directed version of MLA, a nearly-logarithmic lower bound has been established [19].

Also note that, currently, no separation is known between the deadlines and delay settings, or between deterministic and randomized algorithms.

References

1 Yossi Azar, Arun Ganesh, Rong Ge, and Debmalya Panigrahi. Online Service with Delay. *ACM Trans. Algorithms*, 17(3), 2017. (STOC 2017). doi:10.48550/arXiv.1708.05611.

2 Yossi Azar and Noam Touitou. General framework for metric optimization problems with delay or with deadlines. In *Proceedings of the 60th Annual Symposium on Foundations of Computer Science (FOCS)*, 2019. arXiv:1904.07131.

3 Yossi Azar and Noam Touitou. Beyond tree embeddings – a deterministic framework for network design with deadlines or delay. In *Proceedings of the 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1368–1379, 2020. doi:10.1109/FOCS46700.2020.00129.

4 Marcin Bienkowski, Martin Böhm, Jaroslaw Byrka, Marek Chrobak, Christoph Dürr, Lukas Folwarczny, Lukasz Jez, Jiri Sgall, Nguyen Kim Thang, and Pavel Vesely. Online Algorithms for Multi-Level Aggregation. In *Proceedings of the 24th Annual European Symposium on Algorithms (ESA)*, volume 57, page 12(17), 2016. doi:10.4230/LIPIcs.ESA.2016.12.

5 Marcin Bienkowski, Jaroslaw Byrka, Marek Chrobak, Łukasz Jeż, Dorian Nogneng, and Jiří Sgall. Better approximation bounds for the joint replenishment problem. In *Proceedings of the 2014 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 42–54, 2014. doi:10.1137/1.9781611973402.4.

6 Marcin Bienkowski, Artur Kraska, and Paweł Schmidt. Online service with delay on a line. In *25th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 237–248, June 2018. doi:10.1007/978-3-030-01325-7_22.

7 Sébastien Bubeck, Christian Coester, and Yuval Rabani. The randomized k-server conjecture is false! In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC)*, pages 581–594, 2023. doi:10.1145/3564246.3585132.

8 N. Buchbinder, T. Kimbrelt, R. Levi, K. Makarychev, and M. Sviridenko. Online make-to-order joint replenishment model: primal dual competitive algorithms. *Operations Research*, 61(4), 2013. (SODA 2008). doi:10.1287/OPRE.2013.1188.

9 Niv Buchbinder, Moran Feldman, Joseph (Seffi) Naor, and Ohad Talmon. O(depth)-competitive algorithm for online multi-level aggregation. In *Proceedings of the 2017 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1235–1244, 2017. doi:10.1137/1.9781611974782.80.

10 Daniel R. Dooly, Sally A. Goldman, and Stephen D. Scott. TCP dynamic acknowledgment delay (extended abstract): theory and practice. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC)*, pages 389–398, 1998. doi:10.1145/276698.276792.

11 Matthias Englert, Harald Räcke, and Matthias Westermann. Reordering buffers for general metric spaces. *Theory of Computing*, 6(1):27–46, 2010. (STOC 2007). doi:10.4086/TOC.2010.V006A002.

12 Iftah Gamzu and Danny Segev. Improved online algorithms for the sorting buffer problem on line metrics. *ACM Transactions on Algorithms*, 6(1):15:1–15:14, 2009. doi:10.1145/1644015.1644030.

13 Anupam Gupta, Amit Kumar, and Debmalya Panigrahi. Caching with time windows and delays. *SIAM Journal on Computing*, 51(4):975–1017, 2022. doi:10.1137/20M1346286.

14 Anupam Gupta, Amit Kumar, and Debmalya Panigrahi. A hitting set relaxation for k-server and an extension to time-windows. In *Proceedings of the 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 504–515, 2022. doi:10.1109/FOCS52979.2021.00057.

15 Anna R. Karlin, Claire Kenyon, and Dana Randall. Dynamic TCP acknowledgement and other stories about e/(e-1). *Algorithmica*, 36(3):209–224, 2003. (STOC 2001). doi:10.1145/380752.380845.

16 Elias Koutsoupias and Christos H. Papadimitriou. On the k-server conjecture. *Journal of the ACM*, 42(5):971–983, September 1995. doi:10.1145/210118.210128.

17 Predrag Krnetić, Darya Melnyk, Yuyi Wang, and Roger Wattenhofer. The k-Server Problem with Delays on the Uniform Metric Space. In *Proceedings of the 31st International Symposium on Algorithms and Computation (ISAAC)*, volume 181, pages 61:1–61:13, 2020. doi:10.4230/LIPIcs.ISAAC.2020.61.

18 Mark S Manasse, Lyle A McGeoch, and Daniel D Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11(2):208–230, 1990. doi:10.1016/0196-6774(90)90003-W.

19 Noam Touitou. Nearly-Tight Lower Bounds for Set Cover and Network Design with Deadlines/Delay. In *Proceedings of the 32nd International Symposium on Algorithms and Computation (ISAAC)*, volume 212, page 53(16), 2021. doi:10.4230/LIPIcs.ISAAC.2021.53.

20 Noam Touitou. Frameworks for Nonclairvoyant Network Design with Deadlines or Delay. In *Proceedings of the 50th International Colloquium on Automata, Languages, and Programming (ICALP)*, page 105(20), 2023. doi:10.4230/LIPIcs.ICALP.2023.105.

21 Noam Touitou. Improved and deterministic online service with deadlines or delay. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC)*, pages 761–774, 2023. doi:10.1145/3564246.3585107.

22 Noam Touitou. Nearly-Optimal Algorithm for Non-Clairvoyant Service with Delay. In *Proceedings of the 42nd International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 327, pages 74:1–74:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.STACS.2025.74.

A Laziness of Bckt

In this section, we restate and prove the laziness property of the Bckt algorithm.

▶ **Proposition 9.** Bckt *is* 4*-lazy.*

Proof. To prove that Bckt fits the description of a 4-lazy algorithm as defined in Algorithm 1, we need to show the following in the deadlines setting:

- Bckt does not perform any actions while no request has reached its deadline.
- When a single 4-isolated request r reaches its deadline, Bckt serves only r and, if possible, moves closer to r.

Since Bckt waits for a bucket to become full before serving any requests or moving the server, it does not perform any actions while no request has reached its deadline.

In the case where a request at the server's current position becomes critical (i.e., reaches its deadline), Bckt does not initiate a serving phase, but just serves the request without moving. This aligns with the requirements for a 4-lazy algorithm.

Now consider the case where a single 4-isolated request r reaches its deadline at a time t with $s_t^{\text{start}} \neq x_r$ (i.e., the server is not at r when r becomes critical). Let i be the label of the bucket that contains r. Since, in the deadlines setting, no requests other than r have positive delay cost, the critical bucket must be B_i. Bckt serves all requests within the cleaning area $\bigcup_{1\leq j\leq |i|+1}(B_{-j}\cup B_{+j}) \subseteq (s_t^{\text{start}}-4d(r,s_t^{\text{start}}), s_t^{\text{start}}+4d(r,s_t^{\text{start}}))$. Since r is 4-isolated, Bckt serves only r, as required for a 4-lazy algorithm. The next server position s_t^{end} is the closest point of the critical bucket B_i (the point $\pm 2^{|i|-1}$), which satisfies $d(s_t^{\text{end}}, r) < d(s_t^{\text{start}}, r)$. Therefore, the server moves closer to the critical request if possible. ◀

B Laziness of Touitou

In this section, we restate and prove the laziness property of the Touitou algorithm. In particular, we observe that on line metrics, the level of every request remains at the initial value $-\infty$, and that the resulting simplified algorithm is 16-lazy.

▶ **Lemma 10.** *During a service S of* Touitou*, triggered by a request r with $l_r = -\infty$, only requests $q \in R_t$ with $d(q, s_t^{\text{start}}) < 16d(r, s_t^{\text{start}})$ are eligible for service.*

Proof. Every request q that is eligible for service must satisfy $l_S \geq \bar{l}_q \geq \lceil \log_2 d(q, s_t^{\text{start}}) \rceil$. Given that $l_r = -\infty$, we know that $l_S = \bar{l}_r + 3 = \lceil \log_2 d(r, s_t^{\text{start}}) \rceil + 3$. It follows that

$$d(q, s_t^{\text{start}}) \leq 2^{l_S} = 2^{\lceil \log_2 d(r, s_t^{\text{start}}) \rceil + 3} < 16d(r, s_t^{\text{start}}) \,.$$ ◀

▶ **Lemma 11.** *On a line metric,* Touitou *never increases levels of requests.*

Proof. During a service S, only requests q with $d(s_t^{\text{start}}, q) \leq 2^{l_S}$ are eligible for service. On a line metric, the cost of an optimal Steiner tree connecting any such requests to s_t^{start} is at most $2 \cdot 2^{l_S}$. It follows that the cost of a 2-approximation for the Steiner tree problem on this instance is bounded by $4 \cdot 2^{l_S}$. The loop over eligible requests therefore does not break early, meaning that all eligible requests are selected for service. Since the level of a request is only increased if it is eligible for service but not served, the level of a request is never increased. ◀

▶ **Proposition 12.** Touitou *is* 16*-lazy for inputs on a line metric.*

Proof. As before, we need to show that for inputs on a line metric, the following properties hold:

- TOUITOU does not perform any actions while no request has reached its deadline.
- When a single 16-isolated request r reaches its deadline, TOUITOU serves only r and, if possible, moves closer to r.

TOUITOU only performs actions when a request reaches its deadline during the call to UponDeadline, as required.

Now consider the case where a single 16-isolated request r reaches its deadline. By Lemma 11, we have $l_r = -\infty$. We can therefore apply Lemma 10 to find that in the following service S, only r is eligible for service. Hence, only r will be served.

Consider the case where the server is at the location of r at time t_r. By definition of TOUITOU, the server never moves farther away from the critical request, proving that $d(s^{\text{end}}_{t_r}, r) = 0$. In any other case, since all levels are $-\infty$ (Lemma 11), the adjusted level of each request is dictated by its distance to the server. It follows that the service is primary, and the server moves to r. ◀

Simple, Strict, Proper, and Directed: Comparing Reachability in Directed and Undirected Temporal Graphs

Michelle Döring
Hasso Plattner Institute, University of Potsdam, Germany

Abstract

Temporal graphs model networks whose connections are available only at specific points in time. Several definitional subtleties – whether paths must follow strictly increasing time labels (*strict* vs. *non-strict*), whether adjacent edges cannot appear simultaneously (*proper*), and whether edges are forbidden to appear multiple times (*simple*) – give rise to different temporal graph settings. These distinctions directly impact the definition of temporal reachability, a core concept in temporal graph theory. Casteigts, Corsini, and Sarkar [TCS24] introduced a framework of equivalence notions to compare the expressive power of these settings focusing solely on undirected temporal graphs. In this work, we extend their framework to include the fundamental dimension of *directed* vs. *undirected*.

Our contribution is three-fold. We (1) complete the undirected hierarchy by resolving the two open questions from [TCS24], (2) fully characterize the hierarchy of the directed settings, and (3) compare the directed and undirected settings, showing that directed temporal graphs are strictly more expressive than undirected temporal graphs in terms of reachability.

Our structural results highlight both the limitations and strengths of various temporal graph settings – for example, directed + strict + simple graphs can realize every possible reachability graph, while directed + proper graphs necessarily induce at least one transitive reachability on each directed cycle. We also provide transformation procedures between temporal settings offering practical tools for transferring algorithms and hardness results across models.

2012 ACM Subject Classification Mathematics of computing → Graph theory; Mathematics of computing → Paths and connectivity problems; Networks → Network dynamics

Keywords and phrases temporal graphs, directed graphs, temporal reachability, dynamic networks

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.27

Related Version *Full Version*: `https://arxiv.org/abs/2501.11697`

Funding *Michelle Döring*: German Federal Ministry for Education and Research (BMBF) through the project "KI Servicezentrum Berlin Brandenburg" (01IS22092).

Acknowledgements I warmly thank Arnaud Casteigts for his advice and inspiring discussions, and George Skretas for his steady support, encouragement, and valuable feedback on this work.

Proof of statements marked with (⋆) can be found in the full version of the paper.

1 Introduction

A graph is called *temporal* if its connections change over time. Across disciplines, this concept has been given a myriad of names, including highly dynamic, edge-scheduled, multistage, time-labeled, or time-varying graph (or network). Unlike classical *static* graphs, where every edge is available at all times, temporal graphs restrict the availability of each edge to specific points in time, making them ideal for modeling dynamic interactions and processes. In the context of this paper, a temporal graph is defined as $\mathcal{G} = (V, E, \lambda)$, with a finite set of vertices V, a set of directed or undirected edges $E \subseteq V \times V$, and a labeling function $\lambda\colon E \to 2^{\mathbb{N}}$ assigning a set of time labels to every edge, indicating when the edge is present.

© Michelle Döring;
licensed under Creative Commons License CC-BY 4.0

Reachability is a central concept in temporal graph theory and has been the focus of much of recent research. A vertex u is said to *reach* a vertex v in a temporal graph if there exists a temporal path – a sequence of edges traversed in temporal order – from u to v. Reachability in static graphs is transitive (if x reaches y and y reaches z, then x reaches z), which forms the basis of many classical algorithms and structural results. In temporal graphs transitivity is not guaranteed, which breaks standard algorithmic techniques and leads to the hardness of many computational problems. Reachability plays a role in various problems, including optimal journeys [17, 20, 23], exploration [5, 15], infection spreading [19, 12], connected components [4, 11], flows and separators [3, 16, 22], and temporal spanners [7, 9]. It is therefore critical to understand the structure and limitations of temporal reachability.

The temporal reachability of a graph can be represented by the reachability graph $\mathcal{R}(\mathcal{G})$: a static graph which contains an edge from u to v if and only if a temporal path from u to v exists in $\mathcal{G}$. The study of temporal graphs realizing specific reachability patterns dates back to gossip theory [18, 6], which seeks fully connected temporal graphs using few temporal edges. This was later extended to temporal graph realization problems [21, 1], where one aims to construct a temporal graph satisfying prescribed constraints, such as bounded distances. More recent work has shifted towards understanding the space of **all** reachability graphs that can be realized by a given type of temporal graphs, either from a structural perspective, as in [8], or from a computational complexity perspective [14].

Our work builds on the former line of research, extending the structural analysis of Casteigts, Corsini, and Sarkar (CCS) [8] to directed graphs. Their framework of temporal graph *settings* identifies 3 dimensions within the definition of an undirected temporal graph:
(1) *strict* vs *non-strict* (whether times along a path are strictly increasing or non-decreasing);
(2) *proper* vs *arbitrary* (whether adjacent edges are forbidden to appear at the same time);
(3) *simple* vs *multi-labeled* (whether edges can appear at most once or multiple times).
We extend this framework by introducing a fourth, essential dimension:
(0) *directed* vs *undirected*.
This distinction is fundamental, as it directly shapes the definitions of graph properties and the formulation of computational problems, both in temporal and static graph theory.

Every combination of the four dimensions defines a *temporal graph setting* which we denote by its defining properties; for instance, the setting of directed, strict and proper graphs is denoted by directed + strict + proper. These temporal graph classes can be compared under four equivalence relations introduced by CCS, which capture progressively weaker notions of similarity between two graphs $\mathcal{G}$ and $\mathcal{H}$. Such a comparison is established by either providing a transformation that maps every graph in one class to an equivalent graph in the other, or by providing a *separating structure* – a graph that cannot be transformed under the given notion. The four equivalence notions are:

- *Bijective equivalence*: there is a one-to-one matching between all temporal paths in $\mathcal{G}$ and those in $\mathcal{H}$ such that the matched paths visit the same vertices in the same order.
- *Support equivalence*: every path in $\mathcal{G}$ corresponds to a path in $\mathcal{H}$ that visits the same vertices in the same order, and vice versa. Thus, $\mathcal{G}$ and $\mathcal{H}$ must have the same footprint.
- *Reachability equivalence*: both graphs induce the same *reachability graph* $\mathcal{R}(\mathcal{G}) = \mathcal{R}(\mathcal{H})$
- *Induced-reachability equivalence*: $\mathcal{H}$ may have additional vertices, but the subgraph of $\mathcal{R}(\mathcal{H})$ induced by $V(\mathcal{G})$ must be the same as $\mathcal{R}(\mathcal{G})$.

Refer to Figure 1 for an illustration of the first three equivalence notions. In this work, we extend the analysis of temporal settings with respect to support, reachability, and induced-reachability equivalence. Our focus lies primarily on reachability equivalence as it offers a practical middle ground: support equivalence is overly restrictive – transformations can

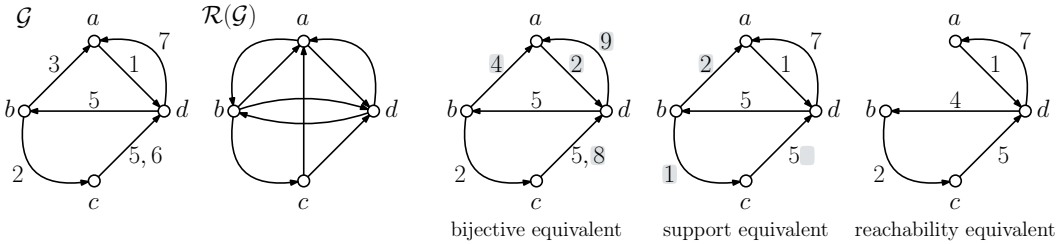


Figure 1 A temporal graph $\mathcal{G}$ and its reachability graph $\mathcal{R}(\mathcal{G})$ under strict reachability (note that c cannot reach b although there exists a non-strict temporal path). On the right are three graphs which are (1) bijective equivalent, (2) support but not bijective equivalent, and (3) reachability but not support equivalent to $\mathcal{G}$. For the former two, differences in the labeling from $\mathcal{G}$ are highlighted in gray. Observe that the rightmost graph is proper and simple.

only relabel edges without modifying the underlying graph; reachability equivalence is more flexible, allowing the addition of edges; induced-reachability allows both edge and vertex additions which is so permissive that all settings collapse into a single class. Since bijective equivalence is even more restrictive than support equivalence, we do not consider this notion.

Results and Technical Overview. Our contribution is three-fold: (1) we resolve the two open questions from [8] in Section 3, thereby completing the hierarchy for undirected temporal settings under reachability equivalence; (2) we fully resolve the directed hierarchy in Section 4 using new separating structures and transformations; (3) we compare the directed and undirected settings in Section 5 to obtain a unified hierarchy. See Figure 2 for an overview of the reachability hierarchies.

To resolve the open questions from CCS, we show that there exists an undirected + non-strict + simple graph whose reachability graph cannot be achieved by an undirected + strict + simple graph. Consequently, these settings are incomparable under reachability equivalence and the undirected reachability hierarchy admits two strands.

We compare each pair of directed temporal graph classes and either provide a transformation or a separating structure proving that no transformation exists. The main difference to the undirected graph classes is that directed + strict + simple is equivalent to directed + strict, both of which can express every reachability graph.

Our directed separating structures are quite different and partially more intricate than the undirected separations provided by CCS. This can be explained as follows. In undirected graphs, adding an edge ab to the footprint implies bidirectional reachability between a and b, which has strong implications on the reachability graph. Therefore, already on short undirected paths the possible footprint for an equivalent graph can be quite restricted. In directed graphs, on the other hand, an edge insertions can introduce specific asymmetric reachabilities. Thus, to construct a graph whose reachability graph cannot be achieved in a specific directed graph class more intricate constructions become necessary.

On the constructive side, we extend the known transformations from [8] to transformations on directed graphs, where the application to undirected graphs emerges as special instances. Beyond this, we introduce a novel transformation called *reachability-dilation*, which replaces each connected component within a snapshot by a bidirected spanning tree. These trees are then labeled using a *pivot-labeling* inspired by [2, 19, 10] and adjusted to ensure properness. This procedure transforms each directed + non-strict graph into a reachability equivalent directed + proper graphs with at most twice the lifetime. Crucially, it also enables a transformation from an undirected to a directed setting, namely from undirected + non-strict + simple to directed + proper + simple.

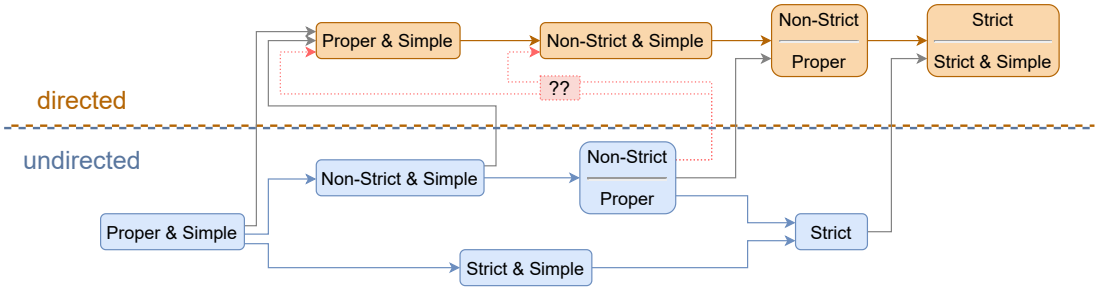


Figure 2 Illustration of the reachability hierarchy among the temporal graph classes. An arrow from (1) to (2) indicates that every graph in (1) can be transformed into a reachability equivalent graph in (2). These transformations are transitive and implied arrows are omitted. The absence of an arrow (unless implied transitively) indicates that no such transformation can exist. If two classes appear in the same box, they induce the same reachability graph. The two red dotted arrows represent the open questions. The top half (orange) shows the directed hierarchy (Section 4) and the bottom half (blue) the undirected hierarchy (Section 3). Their comparison is discussed in Section 5.

2 Preliminaries

A *temporal graph* $\mathcal{G} = (V, E, \lambda)$ consists of a static graph $G = (V, E)$, called the *footprint*, and a labeling function λ. A pair (e, t), where $e \in E$ and $t \in \lambda(e)$, represents a *temporal edge* with *label* t. The set of temporal edges is denoted by $\mathcal{E}$. The *lifetime* τ of $\mathcal{G}$ is the maximum label assigned by λ. The static graph $G_t = (V, E_t)$, where $E_t = \{e \in E \mid t \in \lambda(e)\}$, is called the *snapshot* of $\mathcal{G}$ at time t. A *temporal path* is a sequence of temporal edges $P = \langle (e_i, t_i) \rangle$ where $\langle e_i \rangle$ forms a path in the footprint and the time labels $\langle t_i \rangle$ are non-decreasing. If the time labels $\langle t_i \rangle$ are strictly increasing, P is a *strict* temporal path; otherwise, it is *non-strict*. We denote a temporal path from u to v by $u \rightsquigarrow v$ and say u *can reach* v.

2.1 Reachability and Equivalence Relations

The reachability relation between vertices of a temporal graph $\mathcal{G}$ can be captured by the *reachability graph* $\mathcal{R}(\mathcal{G})$ which is given by the static directed graph (V, E_c) with $(u, v) \in E_c$ if and only if there exists a temporal path from u to v. A graph is *temporally connected* if all vertices can reach each other by at least one path, i. e., $\mathcal{R}(\mathcal{G})$ is a complete directed graph. A *temporally connected component* of $\mathcal{G}$ is a subset of vertices which are temporally connected.

We now define the equivalence relations between temporal graphs in regards to reachability. The definitions are adopted from [8, Section 3]. Let $\mathcal{G}_1 = (V_1, E_1, \lambda_1)$ and $\mathcal{G}_2 = (V_2, E_2, \lambda_2)$ be temporal graphs and denote by $\mathbb{P}(\mathcal{G}_1)$ and $\mathbb{P}(\mathcal{G}_2)$ the sets of all temporal paths in $\mathcal{G}_1$ and $\mathcal{G}_2$, respectively. We say two temporal paths P and P' share the same *support* if they visit the same vertices in the same order, i. e., their underlying static paths are the same.

▶ **Definition 2.1** (Support equivalence). *Two temporal graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ are* support equivalent *if $V_1 = V_2$ and for every temporal path P in either graph, there exists a temporal path P' in the other graph, such that P and P' share the same support.*

▶ **Definition 2.2** (Reachability equivalence). *Two temporal graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ are* reachability equivalent *if $V_1 = V_2$ and $\mathcal{R}(\mathcal{G}_1) \simeq \mathcal{R}(\mathcal{G}_2)$, i. e., the reachability graphs are isomorphic.*

Abusing terminology, we say $\mathcal{G}_1$ and $\mathcal{G}_2$ have the same reachability graph if $\mathcal{R}(\mathcal{G}_1) \simeq \mathcal{R}(\mathcal{G}_2)$.

▶ **Definition 2.3** (induced-Reachability equivalence). *A temporal graph $\mathcal{G}_2$ is* induced-reachability equivalent *to $\mathcal{G}_1$ if $V_1 \subseteq V_2$ and $\mathcal{R}(\mathcal{G}_1) \simeq \mathcal{R}(\mathcal{G}_2)[V_1]$.*

The equivalence notions are sorted by decreasing requirement strength: support equivalence implies reachability equivalence, which in turn implies induced-reachability equivalence.

In this paper, we compare the expressive power of temporal settings with respect to support, reachability, and induced-reachability equivalence. Let $\mathbb{S}$ and $\mathbb{T}$ be two temporal settings, and let $X \in \{S, R, iR\}$ denote one of the three equivalence notions. We say that $\mathbb{S}$ can be *transformed into* $\mathbb{T}$ under X, denoted $\mathbb{S} \rightsquigarrow^X \mathbb{T}$, if for every temporal graph $\mathcal{G} \in \mathbb{S}$ there exists a graph $\mathcal{H} \in \mathbb{T}$ such that $\mathcal{G}$ and $\mathcal{H}$ are equivalent under X. If no such transformation exists, then there is a *separating structure* (or *separation*) with respect to X: a graph $\mathcal{G} \in \mathbb{S}$ for which no graph in $\mathbb{T}$ is X-equivalent to $\mathcal{G}$. We say that $\mathbb{T}$ is *strictly more expressive* than $\mathbb{S}$ under X if $\mathbb{S} \rightsquigarrow^X \mathbb{T}$ but $\mathbb{T} \not\rightsquigarrow^X \mathbb{S}$; the settings are *incomparable* if $\mathbb{S} \not\rightsquigarrow^X \mathbb{T}$ and $\mathbb{T} \not\rightsquigarrow^X \mathbb{S}$, and they are *equivalent* if $\mathbb{S} \rightsquigarrow^X \mathbb{T}$ and $\mathbb{T} \rightsquigarrow^X \mathbb{S}$.

2.2 Settings: (Un)Directed, (Non-)Strict, Simple, and Proper

We identify four key dimensions for defining the settings of a temporal graph $\mathcal{G}$, following [8]:

Directed vs. Undirected A temporal graph is directed (resp. undirected) if its footprint is a directed (resp. undirected) graph.

Strict vs. Non-strict A temporal graph is strict (resp. non-strict) if the reachability graph considers only strict (resp. non-strict) temporal paths.

Proper vs. Arbitrary A temporal graph is *proper* if no two edges incident to the same vertex share a time label.

Simple vs. Multi-labeled A temporal graph is *simple* if every edge is assigned exactly one time label.

We denote the class of temporal graphs belonging to, for instance, the directed and strict setting, as directed + strict, and if $\mathcal{G}$ belongs to the class of graphs defined by this setting we write $\mathcal{G} \in$ directed + strict. For brevity, we will use the shorthand D + ... and UD + ... to denote directed and undirected settings, respectively.

Observe that strict + proper, non-strict + proper, and proper are equivalent: in a proper temporal graph, adjacent edges do not share a time label, so all temporal paths are inherently strict. Thus, we omit the "(non-)strict" when referring to a proper setting. As a result, we consider 12 settings in total: 6 directed and 6 undirected.

Moreover, by definition, any setting $\mathbb{S}$ admits a support-preserving transformation from its restriction to *proper* or to *simple*; that is, $\mathbb{S}$ + proper $\rightsquigarrow^S \mathbb{S}$ and $\mathbb{S}$ + simple $\rightsquigarrow^S \mathbb{S}$ via the identity mapping.

3 Undirected Hierarchy

In this section, we give a brief overview of the results of CCS for undirected temporal graphs and address the two open questions posed in [8]. CCS established three key transformations: *Dilation*, which transforms every UD + non-strict graph into a support equivalent UD + proper graph; *Saturation*, which transforms every UD + non-strict graph into a reachability equivalent UD + strict graph; and *Semaphore*, which transforms every UD + strict graph into an induced-reachability equivalent UD + proper + simple graph. For the remaining setting pairs, CCS proved the existence of separating structures, built from paths or triangles, containing 3 to 5 vertices.

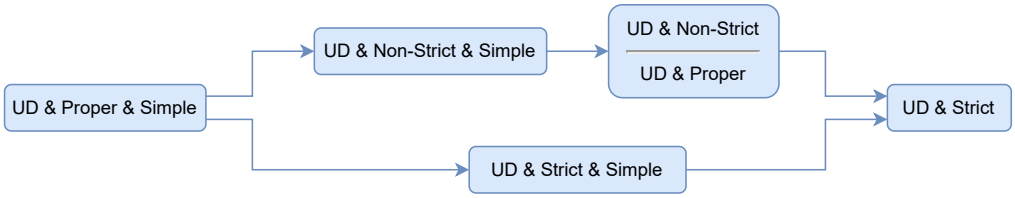
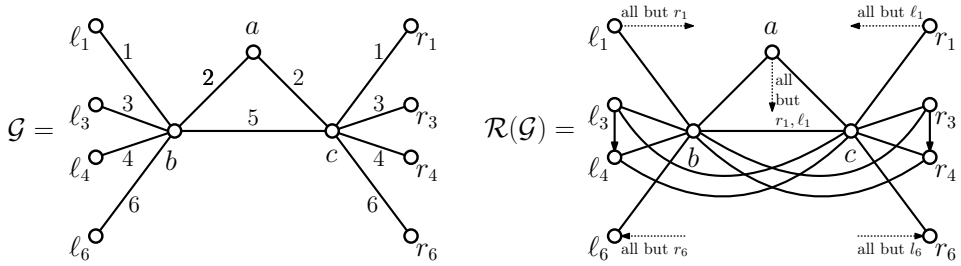


Figure 3 The reachability hierarchy of the undirected temporal graph classes. The undirected hierarchy under support equivalence is identical. Under induced-reachability equivalence, all undirected graph classes are equivalent.

Two questions were left open: whether UD + non-strict can be transformed into UD + strict + simple, and whether UD + non-strict + simple can be transformed into UD + strict + simple. We provide a separating structure for the latter, thereby answering both questions.

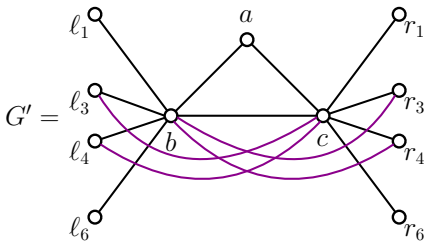
▶ **Theorem 3.1** (UD + non-strict + simple $\not\rightsquigarrow^R$ UD + strict + simple)**.** *There exists a graph* $\mathcal{G} \in$ UD + non-strict + simple *such that there is no* $\mathcal{H} \in$ UD + strict + simple *with* $\mathcal{R}(\mathcal{G}) = \mathcal{R}(\mathcal{H})$.

Proof. Consider the following temporal graph $\mathcal{G}$ in the UD + non-strict + simple setting (left) and the corresponding reachability graph (right). For readabilities sake, the dotted



edges adjacent to $\ell_1, r_1, a, \ell_6, r_6$ indicate incoming or outgoing edges that either connect to all vertices, or connect to all vertices except those explicitly excluded (specified next to the edge). For the sake of contradiction, let $\mathcal{H}$ be a temporal graph in the UD + strict + simple setting whose reachability graph is isomorphic to that of $\mathcal{G}$.

First, observe that there are only four undirected edges apart from the original edges of $\mathcal{G}$ in the reachability graph. That means, that the footprint of $\mathcal{H}$ has to be isomorphic to a subgraph of the following graph G', with the four edges in $G' \setminus G$ highlighted in purple. We



first prove an intermediate statement regarding the purple edges.

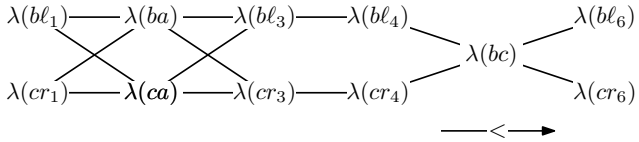
▷ Claim 1. The purple edge $c\ell_3$ is in the footprint of $\mathcal{H}$ if and only if no black edge from $\{cr_3, cr_4\}$ is in the footprint. The same holds for $c\ell_4$ and $\{cr_3, cr_4\}$, for br_3 and $\{b\ell_3, b\ell_4\}$, and for br_4 and $\{b\ell_3, b\ell_4\}$.

Proof of Claim 1. We proof the claim for $c\ell_3$ and $\{cr_3, cr_4\}$; the other three cases follow by a symmetric argument. Assume $c\ell_3$ and cr_3 are in the footprint of $\mathcal{H}$. Then ℓ_3 and r_3 have distance two in the footprint and by [8, Lemma 1], either of them must reach the other. But since $\ell_3 r_3 \notin E(\mathcal{R}(\mathcal{G}))$ and $r_3\ell_3 \notin E(\mathcal{R}(\mathcal{G}))$, this yields a contradiction. The same argument holds for cr_4 in the footprint of $\mathcal{H}$, by [8, Lemma 1] and $\ell_3 r_4, r_4\ell_3 \notin E(\mathcal{R}(\mathcal{G}))$. ◁

This leaves two cases for the footprint of $\mathcal{H}$: either we add no purple edge and H is isomorphic to G, or we add at least one purple edge and delete the corresponding black edges. For the former, we show that there is no simple labeling yielding the desired reachabilities for $\mathcal{H}$.

▷ Claim 2. If the footprint of $\mathcal{H}$ is isomorphic to the footprint of $\mathcal{G}$, then there can be no simple labeling achieving the same reachabilities in the strict setting.

Proof of Claim 2. We show the claim by proving that the chronological order of the edges, i. e., corresponding time labels, cannot change without changing the reachability in the strict setting. Since $\mathcal{H}$ is simple, we also cannot add labels. The chronological order of the edges in $\mathcal{G}$ can be illustrated by the following partially ordered set.



Now, since only ℓ_1, r_1, b, c can reach a, we infer that $\lambda(ba)$ and $\lambda(ca)$ must be strictly smaller than $\lambda(b\ell_3), \lambda(cr_3), \lambda(b\ell_4), \lambda(cr_4), \lambda(b\ell_6), \lambda(cr_6)$, but do not have to be the same. Furthermore, since ℓ_3 can reach ℓ_4 but not vice versa, we have $\lambda(b\ell_3) < \lambda(b\ell_4)$. Analogously, we have $\lambda(cr_3) < \lambda(cr_4)$. Now, for r_3, r_4 to reach b, and ℓ_3, ℓ_4 to reach c, we need $\lambda(b\ell_4) < \lambda(bc)$ and $\lambda(cr_4) < \lambda(bc)$. Lastly, since everyone can reach ℓ_6 and r_6, we know that $\lambda(b\ell_6)$ and $\lambda(cr_6)$ must be greater than all other labels. Given all this, the only way for ℓ_1 to reach everyone but r_1, and vice versa, is by making $\lambda(b\ell_1)$ and $\lambda(cr_1)$ strictly smaller than all other labels. Without the analysis about the ordering of bc before, there would also be other ways for achieving the reachabilities of ℓ_1 and r_1.

This yields exactly the same chronological order for $\mathcal{H}$ as it was the case for $\mathcal{G}$. Now, since we cannot change the relative order of the time labels, $\lambda(ba)$ or $\lambda(ca)$ has to change to enable the reachabilities of ℓ_1 and r_1 in the strict setting. However, if $\lambda(ba) < \lambda(ca)$ then r_1 cannot reach ℓ_3 and ℓ_4, and if $\lambda(ba) > \lambda(ca)$ then ℓ_1 cannot reach r_3 and r_4. Therefore, such a labeling is not possible in the UD + strict + simple setting. ◁

Therefore, $\mathcal{R}(\mathcal{G})$ is not achievable for $\mathcal{H}$ if its footprint is isomorphic to G. So, assume that at least one purple edge is added, without loss of generality, $c\ell_3$, and the corresponding black edges cr_3 and cr_4 are removed. To preserve the reachabilities of r_3 and r_4, they must remain connected to the graph, which can only be done by adding br_3 and br_4. This, however, implies that we need to remove the corresponding black edges $b\ell_3$ and $c\ell_4$. As a result, ℓ_4 becomes disconnected, and we are forced to add $c\ell_4$. Thus, adding even one purple edge to the footprint requires adding all four purple edges. However, adding all purple edges and deleting the corresponding black edges results in a footprint that is isomorphic to G, with only ℓ_3 and ℓ_4 swapping places with r_3 and r_4, respectively. Thus, by Claim 2, there is no simple labeling for $\mathcal{H}$ yielding the desired reachability graph. ◀

Combining the results of [8] with Theorem 3.1, we obtain a two-strand hierarchy for undirected reachability, which is illustrated in Figure 3.

4 Directed Reachability

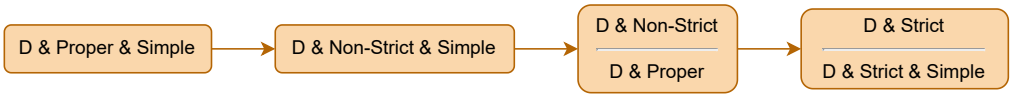


Figure 4 The directed hierarchy under reachability equivalence. Under support equivalence, the directed hierarchy is analogous to the undirected hierarchy (Figure 3). Under induced-reachability equivalence, all directed graph classes are equivalent.

In this section, we resolve the equivalence hierarchy for directed temporal settings. We begin with a key structural observation and then present our results in three parts: reachability separations (Section 4.1), support-only separations (Section 4.2), and transformations between settings (Section 4.3) that preserve support, reachability, or induced-reachability. These results collectively resolve the directed hierarchy, as illustrated in Figure 4.

We begin with a structural limitation which is basis of several of our separations: In non-strict or proper graphs, directed cycles inevitably introduce additional reachabilities.

▶ **Lemma 4.1.** *There exists no labeling of a directed cycle of length at least 3, such that reachability graph of the corresponding* D + non-strict *temporal graph contains only the cycle.*

Proof. Assume towards contradiction that there exists a temporal graph $\mathcal{G} = (G, \lambda)$ such that the footprint G is a directed cycle on the vertices $v_1, \ldots, v_n$, and $\mathcal{R}(\mathcal{G}) = G$. It must hold that $\lambda(v_i, v_{i+1}) > \lambda(v_{i+1}, v_{i+2})$ for all subsequent edges around the cycle, as otherwise v_i can reach v_{i+2} via v_{i+1}. Following this argument, we get $\lambda(v_1, v_2) > \cdots > \lambda(v_{n-1}, v_n) > \lambda(v_n, v_1) > \lambda(v_1, v_2)$, which is a contradiction. ◀

Note that such a labeling can be easily achieved in the D + strict setting: assigning the same label (e.g., 1) to every edge in the cycle avoids transitive paths, as illustrated in Figure 5.

A second structural observation reveals an inherent limitation of specifically the D + proper setting when trying to achieve complete pairwise reachability among n vertices. The claim basically follows from arguments in gossip theory [19] and a short proof can be found in the long version of this paper.

▶ **Lemma 4.2.** *Any* D + proper *temporal graph on $n > 2$ vertices whose reachability graph is a clique (all vertices are pairwise connected) has to contain at least $n + 1$ temporal edges.*

4.1 Reachability separations

The primary separating structure is the directed triangle with the same label on every edge. For this graph in the D + strict + simple setting, there is no reachability equivalent graph in the D + non-strict setting (see Figure 5). This follows directly from Lemma 4.1.

▶ **Lemma 4.3** (D + strict + simple $\not\rightsquigarrow^R$ D + non-strict)**.** *There exists a graph $\mathcal{G} \in$* D + strict + simple *such that there is no $\mathcal{H} \in$* D + non-strict *with $\mathcal{R}(\mathcal{H}) = \mathcal{R}(\mathcal{G})$.*

Recall that the restriction of a setting $\mathbb{S}$ to simple or to proper implies a transformation under support equivalence to $\mathbb{S}$. Hence, Lemma 4.3 also establishes D + strict (+ simple) $\not\rightsquigarrow^R$ D + proper (+ simple) and D + strict (+ simple) $\not\rightsquigarrow^R$ D + non-strict (+ simple).

Next, we separate D + proper (and thus D + non-strict) from D + non-strict + simple (see Figure 6), demonstrating that more than one label per edge is required to capture the full range of directed non-strict reachability. As before, we utilize the structure of a directed triangle by constructing a cycle with two overlapping triangles in the reachability graph. The proof can be found in the extended paper.

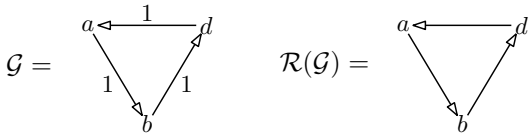


Figure 5 Temporal graph $\mathcal{G} \in$ D + strict + simple (left) and its reachability graph (right). There exists no reachability equivalent graph in the D + non-strict setting.

▶ **Lemma 4.4** ((⋆) D + proper $\not\rightsquigarrow^R$ D + non-strict + simple)**.** *There exists a graph* $\mathcal{G} \in$ D + proper *such that there is no* $\mathcal{H} \in$ D + non-strict + simple *with* $\mathcal{R}(\mathcal{H}) = \mathcal{R}(\mathcal{G})$.

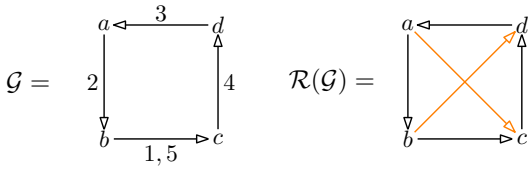


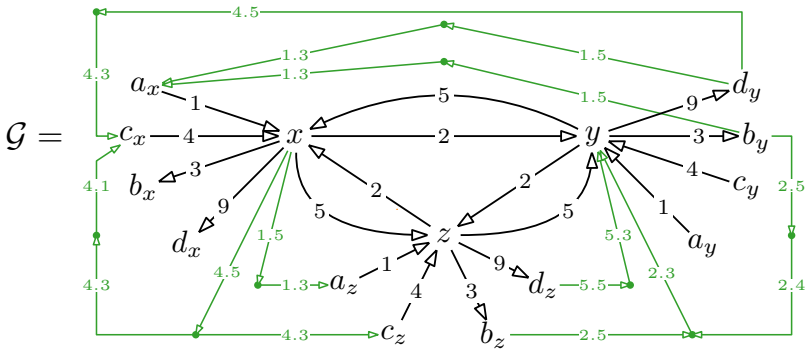
Figure 6 Temporal graph $\mathcal{G} \in$ D + proper (left) and the corresponding reachability graph (right). There exists no reachability equivalent graph in the D + non-strict + simple setting.

Note that this also implies (i) D + non-strict $\not\rightsquigarrow^R$ D + non-strict + simple, (ii) D + proper $\not\rightsquigarrow^R$ D + proper + simple, and (iii) D + non-strict $\not\rightsquigarrow^R$ D + proper + simple.

We now present the final reachability separation, which proves that D + proper + simple is a proper subset of D + non-strict + simple. This separation uses a more intricate construction: three vertices x, y, z are fully connected at two distinct time steps, forming two cliques. Meanwhile, each vertex a_i and c_i (for $i \in \{x, y, z\}$) must traverse these cliques to reach every b_j and d_j with $j \in \{x, y, z\} - i$. Crucially, this traversal has to happen at different time steps so that a_i reaches both b_j and d_j, while c_i reaches only d_j. To enforce this double traversal via x, y, z, additional directed paths are used to forbid certain edges (shortcuts) in the footprint. These paths are constructed symmetrically for each combination of x, y, z and block cycles in the footprint relying on Lemma 4.1.

▶ **Lemma 4.5** (D + non-strict + simple $\not\rightsquigarrow^R$ D + proper + simple)**.** *There exists a graph* $\mathcal{G} \in$ D + non-strict + simple *such that there is no* $\mathcal{H} \in$ D + proper + simple *with* $\mathcal{R}(\mathcal{H}) = \mathcal{R}(\mathcal{G})$.

Proof. Consider the following temporal graph $\mathcal{G}$ in the D + non-strict + simple setting.



For sake of readability, the following edges have been omitted in the illustration, with each set having one representative shown in the illustration in green. Let $C = \{x, y, z\}$.

- $E_1 = \{(i, h_i^{a_j}, 1.5), (h_i^{a_j}, a_j, 1.3), (i, h_i^{c_j}, 4.5), (h_i^{c_j}, c_j, 4.3) \colon i \in C \text{ and } j \in C \setminus \{i\}\}$;
- $E_2 = \{(b_j, h_{b_j}^i, 2.5), (h_{b_j}^i, i, 2.3), (d_j, h_{d_j}^i, 5.5), (h_{d_j}^i, i, 5.3) \colon i \in C \text{ and } j \in C \setminus \{i\}\}$;
- $E_3 = \{(b_j, h_{b_j}^{a_i}, 1.5), (h_{b_j}^{a_i}, a_i, 1.3), (d_j, h_{d_j}^{a_i}, 1.5), (h_{d_j}^{a_i}, a_i, 1.3), (d_j, h_{d_j}^{c_i}, 4.5), (h_{d_j}^{c_i}, c_i, 4.3) \colon$ $i \in C \text{ and } j \in C \setminus \{i\}\}$.
- $E_4 = \{(b_i, s, 2.7), (s, h_{b_j}^i, 2.5), (h_i^{c_j}, s, 4.3), (s, c_j, 4.1) \colon i \in C \text{ and } j \in C \setminus \{i\}\}$;

We refer to the vertices h_α^β as *helper* vertices from α to β and to $\{x, y, z\}$ as the *center* vertices. The edges in E_1, E_2, E_3, and E_4 form paths of length two in the footprint and the time labels are chosen such that they do not form temporal paths. Specifically, for $i \in \{x, y, z\}$ and $j \in \{x, y, z\} \setminus \{i\}$, E_1 forms paths from i to a_j and c_j; E_2 forms paths from b_i and d_i to j; E_3 forms paths from b_i and d_i to a_j and c_j; and E_4 forms paths from b_i to the helper vertices of b_j, and from the helper vertices of c_j to c_i.

For the sake of contradiction, let $\mathcal{H}$ be a temporal graph in the D + proper + simple setting whose reachability graph is isomorphic to that of $\mathcal{G}$. Observe that the center vertices form a strongly connected component in G_2 and in G_5, and recall from Lemma 4.2 that in the D + proper setting, we need at least four temporal edges to fully connect three vertices. Further, observe the following reachabilities in $\mathcal{G}$:

- x, y, z reach every b_i and d_i with $i \in \{x, y, z\}$;
- for $i \in \{x, y, z\}$, every a_i reaches x, y, z, b_j and d_j with $j \in \{x, y, z\}$ and
- every c_i reaches x, y, z and d_j for $j \in \{x, y, z\}$;
- every helper vertex h_α^β can reach the vertex β, but not α. Conversely, α can reach h_α^β, but not β. However, β can reach α.

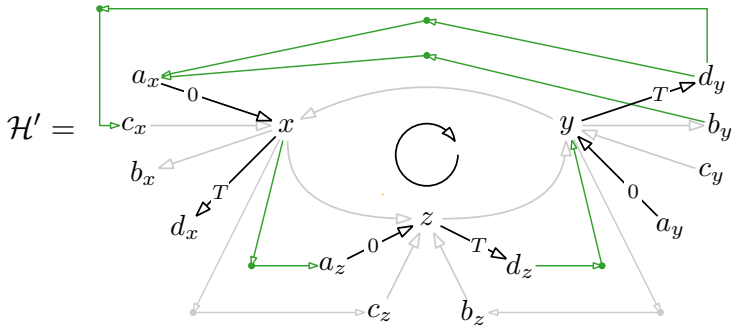
First, we show that the construction of the helper vertices h_β^α forbids the direct edges (β, α) in $\mathcal{H}$ for all but (c_j, i) and (i, b_j).

▷ Claim 1. In $\mathcal{H}$, there can be no edge from $F = \{(a_j, i), (i, d_j), (a_i, b_j), (a_i, d_j), (c_i, d_j), (h_{b_j}^i, b_i), (c_j, h_i^{c_j}) \colon i, j \in \{x, y, z\},\ j \neq i\}$.

Proof of Claim 1. Recall that any labeling of a directed triangle in a non-strict or proper setting leads to at least one transitive reachability (Lemma 4.1), and observe that every direct edge (β, α) would form a directed triangle with the corresponding green edges $(\alpha, h_\alpha^\beta), (h_\alpha^\beta, \beta)$. Let $(\beta, \alpha) \in F$. Observe that h_α^β reaches β, but reaches no other vertex that also reaches β. Thus, the direct edge (h_α^β, β) must be included in the footprint H of $\mathcal{H}$. Furthermore, h_α^β is reached by α but by no other vertex that α can reach. Hence, the direct edge (α, h_α^β) must also be in $E(H)$. As a result, $(\beta, \alpha) \notin E(H)$, since $\alpha, \beta, h_\alpha^\beta$ form a directed triangle in $\mathcal{R}(\mathcal{G})$, which is impossible if H contains the edges of the triangle. ◁

From Claim 1 follows that for a_i to reach d_j in $\mathcal{H}$, no shortcut edge (a_i, j), (i, d_j), or (a_i, d_j) can be used. Additionally, a reaches no vertex that also reaches d_j in $\mathcal{G}$. Thus, a_i must use a temporal path with support $(a_i, i, \ldots, j, d_j)$ in $\mathcal{H}$, and $(a_i, i), (j, d_j) \in E(H)$. As this holds for all i and j, there must exist paths between all center vertices, ensuring that each a_i reaches x, y and z via its respective vertex i. After these paths, there must also be an edge from x, y and z to their respective d_j. The following graph $\mathcal{H}'$ summarizes the current information on $\mathcal{H}$, with some large time step T. Black and green edges must be present in $\mathcal{H}$, while for gray edges no information is yet available. Next, we consider the b and c vertices. Note that the construction does not forbid the direct edges (i, b_j) and (c_j, i) for $i \neq j$ in $\mathcal{H}$:

- $h_{b_j}^i$ is reached only by b_j, requiring $(b_j, h_{b_j}^i) \in E(H)$. However, $h_{b_j}^i$ reaches x, y, z, d_x, d_y, d_z, and b_i via i. Consequently, we could add $(h_{b_j}^i, k)$ for any $k \in \{x, y, z\}$.
- $h_i^{c_j}$ only reaches c_j, requiring $(h_i^{c_j}, c_j) \in E(H)$. Moreover, $h_i^{c_j}$ is reached by x, y, z, a_x, a_y, a_z, and c_i via i. Consequently, we could add $(k, h_i^{c_j})$ for any $k \in \{x, y, z\}$.



While we cannot establish Claim 1 for (i, b_j) or (c_j, i), we can prove the following slightly weaker result. For each b_i, there is exactly one center vertex (out of x, y, z) that has an edge to b_i, while the other two center vertices do not. Similarly, for each c_i, there is exactly one center vertex that c_i has an edge to, while it has no edges to the remaining center vertices. We begin by proving this for the b vertices.

▷ Claim 2. There exists a bijective mapping $f\colon \{x, y, z\} \to \{x, y, z\}$ so that $(f(i), b_i) \in E(H)$ and $(k, b_i) \notin E(H)$ for all $k \in \{x, y, z\} \setminus \{f(i)\}$.

Proof. By Claim 1, a_i cannot reach b_j in $\mathcal{H}$ via the direct edge (a_i, b_j). Therefore, a_i must traverse a temporal path starting at i. Since b_j can only be reached by the center vertices x, y, z and a_x, a_y, a_z, it follows that (x, b_j), (y, b_j) or (z, b_j) must be in $\mathcal{H}$.

If $(i, b_i) \in E(H)$ and the corresponding helper edges are included in $\mathcal{H}$, the claim holds with f defined as the identity mapping. Thus, it remains to prove the claim for the case where $(i, b_i) \notin E(H)$ for at least one $i \in \{x, y, z\}$. Without loss of generality, assume $(z, b_z) \notin E(H)$; the remaining cases follow by symmetry of the construction.

From $(z, b_z) \notin E(H)$ follows $(y, b_z) \in E(H)$ or $(x, b_z) \in E(H)$ to ensure that b_z is reachable. If $(y, b_z) \in E(H)$, then $(h^y_{b_z}, y) \notin E(H)$, as otherwise, $y, b_z, h^y_{b_z}$ would form a directed triangle in both H and $\mathcal{R}(\mathcal{G})$, which is impossible in a simple + proper graph. Consequently, $h^y_{b_z}$ must have an edge to x or to z. The same reasoning applies if $(x, b_z) \in E(H)$, implying $(h^x_{b_z}, x) \notin E(H)$ and requiring $h^x_{b_z}$ to have an edge to y or to z.

Now, observe that the helper vertices cannot have edges to arbitrary center vertices. Let the center vertex $i \in \{x, y, z\}$ have incoming edges from two helper vertices with different out-neighbors in $\mathcal{G}$, for example, h^y_- with an edge to y in $\mathcal{G}$, and h^z_- with an edge to z in $\mathcal{G}$. These two vertices have incompatible reachabilities: h^y_- must reach b_y (but not b_z), and h^z_- must reach b_z (but not b_y). Thus, at most one of the two can reach their respective b vertex via i. If both h^y_- and h^z_- had temporal paths to their b vertex starting in i, they could not depart i at the same time, as this would result in both reaching b_y and b_z. Therefore, one temporal path must occur earlier than the other. However, the helper vertex taking the earlier path could also take the later path and would reach both b_y and b_z. Consequently, one of the helper vertices must reach its respective b vertex via a different center vertex.

This implies that, to satisfy the reachabilities of the helper vertices, all $h^i_{b_j}$ for a fixed i must reach b_i via the same center vertex. Consequently, there exists a mapping $f\colon \{x, y, z\} \to \{x, y, z\}$ such that $(h^i_{b_j}, f(i)) \in E(H)$.

Consider the center vertex y and assume $f^{-1}(y) = x$. According to this mapping, y has incoming edges from $h^x_{b_y}$ and $h^x_{b_z}$. If $(y, b_y) \in E(H)$, then $h^x_{b_y}, y, b_y$ would form a triangle in both H and $\mathcal{R}(\mathcal{H})$ – a contradiction. Similarly, if $(y, b_z) \in E(H)$, then $h^x_{b_z}, y, b_z$ would form a triangle in both H and $\mathcal{R}(\mathcal{H})$. Thus, we conclude that only $(y, b_x) \in E(H)$. The same holds for any other center vertex and any possible mapping f.

Therefore, there exists a bijective mapping f on the center vertices such that $(f(i), b_i) \in E(H)$ for all $i \in \{x, y, z\}$ and $(k, b_i) \notin E(H)$ for all $k \in \{x, y, z\} \setminus \{f(i)\}$. ◁

The proof for the c vertices follows analogously to that of the b vertices. Here, every occurrence of b_i is replaced with c_i, and the direction of each mentioned edge is reversed. Instead of two helper edges pointing toward the same center vertex leading to a contradiction, the contradiction arises from two helper edges pointing away from the same center vertex. This reversal of direction results in the same type of mapping, ensuring that each c_i has an edge to exactly one center vertex, with no edges to the others. The reasoning and conclusions remain identical, making a separate proof unnecessary.

▷ Claim 3. There exists a bijective mapping $g\colon \{x, y, z\} \to \{x, y, z\}$ so that $(c_i, g(i)) \in E(H)$ and $(c_i, k) \notin E(H)$ for all $k \in \{x, y, z\} \setminus \{g(i)\}$.

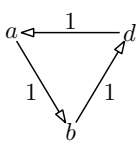
We will now draw the final conclusion, proving that $\mathcal{H}$ can have no proper + simple labeling. Let $i \in \{x, y, z\}$ and f and g be bijections as in Claim 2 and Claim 3. The labeling of $(a_i, i), (c_{g(i)}, i), (i, b_{f(i)}), (i, d_i)$ must ensure that a_i reaches both $b_{f(i)}$ and d_i, while $c_{f(i)}$ reaches only d_i. Consequently, $\lambda(a_i, i) < \lambda(i, b_{f(i)}) < \lambda(i, c_{g(i)}) < \lambda(i, d_i)$. Now, $c_{g(i)}$ has to reach every $d_j, j \in \{x, y, z\}$. Since no shortcut edge can be used, $c_{g(i)}$ must traverse a temporal path with support $(c_{g(i)}, i, \ldots, j, d_j)$. This traversal must happen after $\lambda(c_{g(i)}, i)$, and, consequently, after $\lambda(i, b_{f(i)})$. Thus, the a and the c vertices must travel at different times, requiring an x, y, z-clique to exist at two different points in time. Each of these requires four temporal edges between x, y, and z in a proper graph; eight in total. However, at most six temporal edges can exist between x, y, z in a simple graph, a contradiction. ◀

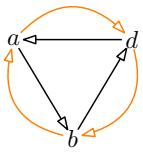
4.2 Support separations

We present the remaining separations that hold only under support equivalence, namely that there exists a D + non-strict and a D + proper temporal graph for which there is no support equivalent D + strict + simple graph. Note that these separations do not hold under reachability equivalence as we give a transformation in Section 4.3.

▶ **Lemma 4.6** (D + non-strict + simple $\not\to^S$ D + strict + simple)**.** *There exists a graph* $\mathcal{G} \in$ D + non-strict + simple *such that there is no support equivalent graph* $\mathcal{H} \in$ D + strict + simple.

Proof. Consider the following simple temporal graph $\mathcal{G}$ (left) in the non-strict setting and the corresponding reachability graph (right). For the sake of contradiction, let $\mathcal{H}$ be a support equivalent temporal graph in the strict + simple setting.

$\mathcal{G} =$ 

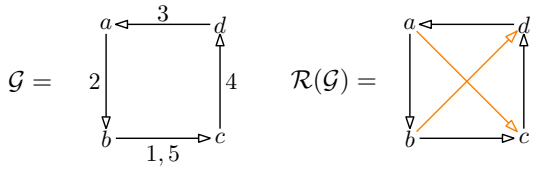
$\mathcal{R}(\mathcal{G}) =$ 

To preserve the support of a to c via b, $\mathcal{H}$ needs $\lambda(a, b) < \lambda(b, c)$. Similarly, for b to reach a via c and c to reach b via a, we get $\lambda(a, b) < \lambda(b, c) < \lambda(c, a) < \lambda(a, b)$, a contradiction. ◀

The separation of D + proper from D + strict + simple, is achieved by the same structure separating D + proper from D + non-strict + simple (Lemma 4.4) and requires only a slightly adjusted argument to fit the support-preserving requirement.

▶ **Lemma 4.7** (D + proper $\not\rightsquigarrow^S$ D + strict + simple)**.** *There exists a graph $\mathcal{G} \in$ D + proper such that there is no support equivalent graph $\mathcal{H} \in$ D + strict + simple.*

Proof. Consider the following temporal graph $\mathcal{G}$ in the D +proper setting (left) and the corresponding reachability graph (right). For the sake of contradiction, let $\mathcal{H}$ be a support equivalent temporal graph in the D +strict+simple setting.



Observe that we require all black edges in $\mathcal{H}$ to preserve the support of the direct reachabilities. Furthermore, there are two transitive reachabilities, namely a reaches c via b and b reaches d via c. Now observe that c not reaching a and d not reaching b imply $\lambda(c,d) > \lambda(d,a) > \lambda(a,b)$. There are two options for the label of (b,c). Either $\lambda(b,c) > \lambda(a,b)$ and $\lambda(b,c) > \lambda(c,d)$, which enables a to reach c via b and avoids a reaching d, or $\lambda(b,c) < \lambda(a,b) < \lambda(c,d)$ which enables b to reach d via c. However, in the first case, a cannot reach c via b and in the second, b cannot reach d via c.

Adding some of the orange edges to $\mathcal{H}$ will not achieve the transitive reachabilities, so there can be no support equivalent labeling. ◀

Since D + proper is a subset of D + strict, this also proves D + strict $\not\rightsquigarrow^S$ D + strict + simple.

4.3 Transformations

In this section, we present four transformations between directed temporal settings. Two of our transformations generalize techniques from [8]: the *dilation* process, extended here as the support-preserving *support-dilation*, and the *semaphore* construction, adapted for directed induced-reachability equivalence. Additionally, we introduce two new transformations: a more efficient reachability-preserving variant called *reachability-dilation*, and the following transformation that maps any directed graph to the strict + simple setting.

▶ **Observation 4.8** (D + * $\rightsquigarrow^R$ D + strict + simple)**.** *Given a directed temporal graph $\mathcal{G}$ in an arbitrary setting, let $\mathcal{H} = (\mathcal{R}(\mathcal{G}), \lambda)$ with $\lambda(e) = 1$ for all $e \in E(\mathcal{R}(\mathcal{G}))$. Then $\mathcal{H} \in$ D + strict + simple with $\mathcal{R}(\mathcal{H}) = \mathcal{R}(\mathcal{G})$.*

Note that this transformation is specific to directed graphs and has no analogue in the undirected temporal graph classes, since reachabilities are not symmetric and directed edges are essential for preserving these dependencies.

4.3.1 Support-Dilation (non-strict $\rightsquigarrow^S$ proper) and Reachability-Dilation (non-strict $\rightsquigarrow^R$ proper)

Before analyzing directed dilation, we compare the structure of the snapshots of non-strict temporal graphs in an undirected and directed setting. In undirected graphs each snapshot consists of 1 to n connected components, each forming a clique in the reachability graph. In contrast, snapshots of directed temporal graphs consist of 1 to n weakly connected components. Each of those can be interpreted as a directed acyclic graph (DAG) with as vertices the strongly connected subgraphs (possibly of size 1) of the weakly connected component connected by directed edges in an acyclic manner. Notably, undirected snapshots

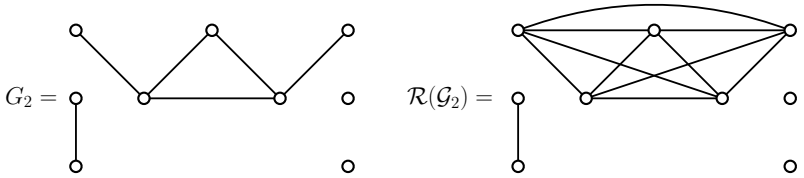


Figure 7 A cutout of some undirected temporal graph at time 2 with four connected components (left) and the corresponding reachability graph with four maximal cliques (right).

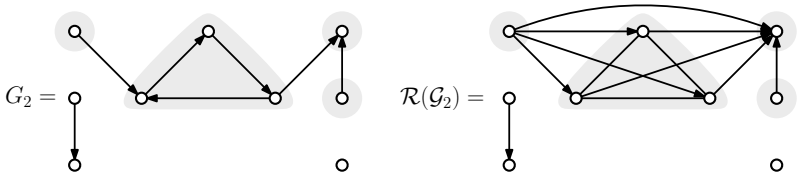


Figure 8 A cutout of some directed temporal graph at time 2 with three weakly connected components (left) and the corresponding reachability graph (right). The strongly connected components of the size-6 weakly connected component are marked in gray.

are a special case of directed ones – an observation that enables us to generalize the undirected dilation technique from [8]. *Dilation* transforms non-strict graphs into proper graphs while preserving support. The process operates on each snapshot individually, duplicating each non-strict path within a connected component of the snapshot, thereby stretching the time step to construct equivalent strict paths. Subsequent snapshots are shifted accordingly. Note that the new lifetime may be as large as $(n-1)\cdot(\Delta+2)\cdot\tau$, where Δ is the maximum degree. For directed graphs, we generalize the dilation process to also handle weakly connected components and call it *support-dilation*.

▶ **Definition 4.9** (Support-Dilation). *Let $\mathcal{G}$ be a temporal graph and consider each snapshot G_t individually. Without loss of generality, we assume $t = 1$ (otherwise, we shift the labels of the earlier snapshots in the construction).*

Every snapshot consists of weakly connected components $\mathbb{W}$ that each can be interpreted as a DAG. We consider each $W \in \mathbb{W}$ separately, so let $D = (\mathbb{S}, E)$ be the DAG representing W. This DAG contains strongly connected components $\mathbb{S}$ (possibly of size 1) as vertices connected by directed edges. Any DAG can be topologically ordered, i. e., the vertices can be arranged as a linear ordering that is consistent with the edge directions. Let ℓ be such an ordering.

First, we order the edges of E lexicographically by the positions of their endpoints in ℓ, i. e., (u, v) precedes (u', v') if $\ell(u) < \ell(u')$, or if $\ell(u) = \ell(u')$ and $\ell(v) < \ell(v')$. Then, in this order, we label the edges in E between *the strongly connected components with distinct time labels starting from 1. Next, in order of ℓ, "dilate" the edges* inside *each strongly connected component as follows (subsequent labels on E are shifted accordingly): Let $S \in \mathbb{S}$ be a strongly connected component and $\alpha = \ell(S)$. First, assign each edge the set of labels $\{\alpha+1, \alpha+2, \ldots, \alpha+k\}$, where k is the longest directed path in S. Now, color the edges of S (ignoring their directions) such that no adjacent edges share the same color. By Vizing's theorem, this requires at most $\Delta + 2$[1] colors, where Δ is the maximum degree of S. Let $c\colon E_S \to [0, \Delta+1]$ be such a coloring and let $0 < \varepsilon < \frac{1}{\Delta+2}$. Now, for every edge $e \in E_S$, add $c(e)\cdot\varepsilon$ to every label of e.*

After processing every snapshot, support-dilation returns the adjusted temporal graph.

[1] We thank the anonymous reviewer of ISAAC25 for pointing out that the usual bound of $\Delta + 1$ from Vizing's theorem for simple graphs does not apply and for kindly pointing to the generalized form for multigraphs which gives the $\Delta + 2$ bound.

Note that the dilation of strongly connected components is a direct analogue of the dilation process of [8], while the main addition is the ordering and temporal shifting of the strongly connected components **within** the DAG representing a weakly connected component.

▶ **Theorem 4.10** (D + non-strict $\rightsquigarrow^S$ D + proper)**.** *Given a* D + non-strict *temporal graph* $\mathcal{G}$*,* support-dilation *transforms* $\mathcal{G}$ *into a* D + proper *graph* $\mathcal{H}$ *such that there is a non-strict temporal path in* $\mathcal{G}$ *if and only if there is a strict temporal path in* $\mathcal{H}$ *with the same support.*

Proof. ($\mathcal{H}$ is proper.) Every snapshot is processed independently and shifted according to changes in previous snapshots. Within each snapshot, the weakly connected components are handled separately, which is valid as they are independent with respect to a proper labeling. Within each weakly connected component, the strongly connected components are processed independently, following the ordering determined by the corresponding DAG. Each strongly connected component is shifted according to changes occurring earlier in the DAG ordering. As argued by [8], the addition of $c(e) \cdot \varepsilon$ to the labels of every edge e according to the coloring c in the strongly connected components ensures that components are labeled "proper"ly. ($\mathcal{H}$ preserves the temporal paths.) Given a weakly connected component in a snapshot G_t of $\mathcal{G}$ considered independently, let $D = (\mathbb{S}, E)$ be the DAG representing it. Without loss of generality, assume $t = 1$. Now, consider some strongly connected component $S \in \mathbb{S}$ at $\alpha = \ell(S)$ in the ordering of D. The longest path in S has length k and in $\mathcal{H}$, every edge of S is assigned all the labels from $\alpha + 1$ to $\alpha + k$. Thus, by the same argument as in [8], for every (non-strict) path of length k' in S, there is a *strict* temporal path in the adjustment of S in $\mathcal{H}$ along the same sequence of edges, going over the same edges but with labels $1, 2, \ldots, k'$ (up to addition of $c(e) \cdot \varepsilon$). Now, since the order of the strongly connected components in a DAG is maintained, temporal paths over multiple strongly connected components in one snapshot are preserved. The same holds for temporal paths spreading over multiple snapshots. ◀

For a reachability-preserving transformation, we propose a more efficient dilation process prioritizing sparsity. Unlike support-dilation, which reconstructs every path within each strongly connected component, *reachability-dilation* preserves the reachabilities in each strongly connected component by replacing it with a bidirected spanning tree. The components are then connected according to the DAG ordering as in the support-dilation process. This approach replaces an edge between u and v at time step t with at most two temporal edges, which results in a lifetime bounded by $2 \cdot \tau$. Refer to Figure 9 for an illustration.

▶ **Definition 4.11** (Reachability-Dilation)**.** *Let* $\mathcal{G}$ *be a temporal graph and consider each snapshot* G_t *individually. Without loss of generality,* $t = 1$*. Let* $D = (\mathbb{S}, E)$ *be the DAG representing a weakly connected component of* G_1*, and* ℓ *an ordering of* D*.*

First, we order the edges of E *lexicographically by the positions of their endpoints in* ℓ *(see Definition 4.9 and label the edges in* E between *the strongly connected components in this order with distinct time labels starting from 1. Second, in the order of* ℓ*, we replace the edges* inside *each strongly connected component as follows (subsequent labels on* E *are shifted accordingly): Let* $S \in \mathbb{S}$*. As* S *is strongly connected, the underlying undirected graph contains a spanning tree* T_S*. It was shown by [10] that one can temporally connect any undirected tree using at most two labels per edge. Slightly adjusting their construction and argument, we temporally connect* T_S *by turning it into a bidirected tree and placing one distinct time label one each directed edge: Choose some vertex of* S *as the arbitrary root. Starting at the leafs, assign distinct, increasing labels to the upwards edge until reaching the root. Now, starting at the root, assign further increasing labels to the downwards edges until reaching every leaf.*

After processing every snapshot, reachability-dilation returns the adjusted temporal graph.

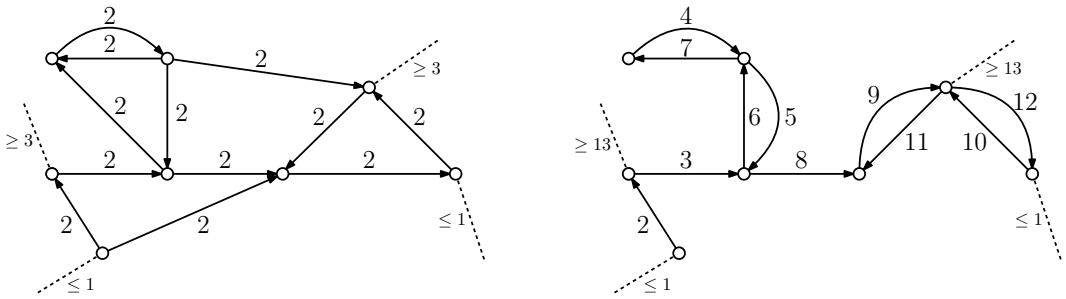


Figure 9 Illustration of the *reachability-dilation* described in Definition 4.11 for some $\mathcal{G}$ with snapshot G_2 (left) and the proper labeling with the bidirected subtrees in the DAG structure (right). The dotted edges represent adjacent earlier and later (shifted in the proper labeling) edges.

In the following, we prove that reachability-dilation is reachability-preserving. But first, we show that the modified spanning tree construction yields a temporally connected graph.

▶ **Lemma 4.12.** *Any bidirected tree with root r, whose labels are assigned as in Definition 4.11 (starting at the leafs, assign distinct, increasing labels to the upwards edge until reaching the root – then starting at the root, assign further increasing labels to the downwards edges until reaching every leaf) is temporally connected.*

Proof. By construction, the upwards directed edges form temporal paths from the leafs to the root r, and the downwards directed edges form temporal paths from r to the leafs. Since all upwards paths are assigned time labels strictly smaller than all labels of the downwards paths, r is a pivot vertex, i. e., there is a time step t such that every vertex of the tree reaches r before t and can be reached by r after t. Thus, the tree is temporally connected. ◀

▶ **Theorem 4.13.** *Given a* D + non-strict *temporal graph $\mathcal{G}$,* reachability-dilation *transforms $\mathcal{G}$ into a* D + proper *temporal graph $\mathcal{H}$ with $\mathcal{R}(\mathcal{G}) = \mathcal{R}(\mathcal{H})$.*

Proof. ($\mathcal{H}$ is proper.) This follows by the same arguments as in Theorem 4.10 and the fact that the spanning tree labeling is proper.
($\mathcal{R}(\mathcal{G}) = \mathcal{R}(\mathcal{H})$.) Given a weakly connected component in a snapshot G_t of $\mathcal{G}$ considered independently, let $D = (\mathbb{S}, E)$ be the DAG representing it. Let $S \in \mathbb{S}$ be a strongly connected component in D. By Lemma 4.12, the edges in S are replaced by a proper, temporally connected bidirected tree spanning all vertices of S. Thus, all vertices in S can reach one another in $\mathcal{H}$ within the extended time interval of S, so the reachability inside S in $\mathcal{H}$ is the same as in $\mathcal{G}$. Now, since the chronological order of the strongly connected components in any DAG is maintained and the footprint of the DAG is the same, temporal reachability over multiple strongly connected components in one snapshot is the same in $\mathcal{H}$ and $\mathcal{G}$. Lastly, since the chronological order of the snapshots is also maintained, the temporal reachability via multiple snapshots is also equivalent. ◀

Reachability-dilation can also be executed on undirected graphs, where it replaces each connected component of every snapshot with an undirected spanning tree. This process will be discussed in more detail in Section 5, Lemma 5.3.

4.3.2 Directed Semaphore: D + strict $\rightsquigarrow^{iR}$ D + simple + proper

We adapt the semaphore technique of [8] to directed graphs. This transformation preserves *induced-reachability* while producing a simple + proper graph by replacing each edge with a path of length 2 to ensure a simple labeling and shifting labels to ensure a proper labeling.

▶ **Definition 4.14** (Semaphore (directed))**.** *Let $\mathcal{G}$ be a directed temporal graph with temporal edges $\mathcal{E}$, let Δ be the maximum degree of the footprint G, and let $0 < \varepsilon < \frac{1}{2(\Delta+1)}$. Consider an edge-coloring c of G using $\Delta + 1$ colors (which exists by Vizing's theorem).*

Replace every directed temporal edge $e = (u, v, t) \in \mathcal{E}$ with an auxiliary vertex $\overrightarrow{uv}$ and two temporal edges $(u, \overrightarrow{uv}, t - c(e) \cdot \varepsilon)$ and $(\overrightarrow{uv}, v, t + c(e) \cdot \varepsilon)$.

▶ **Theorem 4.15** (D + strict $\rightsquigarrow^{iR}$ D + simple + proper)**.** *Given a temporal graph $\mathcal{G} \in$ D + strict, semaphore transforms $\mathcal{G}$ into a* D + simple + proper *temporal graph $\mathcal{H}$ such that there is $\sigma \colon V(\mathcal{G}) \to V(\mathcal{H})$ with $(u, v) \in \mathcal{R}(\mathcal{G})$ if and only if $(\sigma(u), \sigma(v)) \in \mathcal{R}(\mathcal{H})$.*

The proof of Theorem 4.15 is almost identical to that of [8, Theorem 2], so we give only a brief intuition: $\mathcal{H}$ is obviously simple, and proper by the same reasoning as for Theorem 4.10. The reachability equivalence for $V(\mathcal{G})$ follows from the construction: Each directed edge (u, v, t) is subdivided into two parts. The first half is assigned a label less than t, and the second half is assigned a label greater than t (shifted according to the edge coloring). As a result, after traversing from u to v via the subdivided edge, any temporal path reaches v at a time later than any first-half label of any original edge (v, x, t). This ensures that a temporal path can take at most one of the original edges at each time step, and thus is strict.

As for undirected graphs, one can apply dilation and semaphore, to transform any directed setting into D + simple + proper with preserved induced-reachability. Since D + simple + proper is subset of every setting, we conclude the following.

▶ **Corollary 4.16** (D + * $\leftrightsquigarrow^{\text{iR}}$ D + simple + proper)**.** *All directed temporal graph classes are induced-reachability equivalent.*

5 Comparison of Directed and Undirected Reachability

Finally, we compare directed and undirected temporal graph classes with respect to reachability equivalence. Refer back to Figure 2 for an overview.

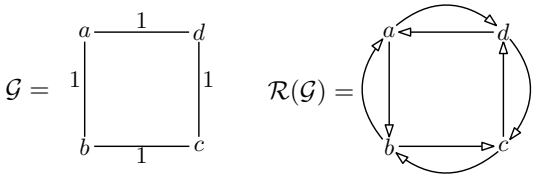
First, observe that directed settings are inherently more expressive than their undirected counterparts: the reachability graph of a single directed edge cannot be realized in any undirected setting under equivalence notions stronger than induced-reachability, whereas each undirected edge can be replaced by two opposing directed edges. This suggests that any graph in an undirected setting UD + x can be transformed into a support equivalent graph in the corresponding directed setting D + x using this simple replacement. For both strict and non-strict settings, this transformation works seamlessly, and the resulting directed graph is simple if and only if the original undirected graph was simple. For proper graphs, however, the opposing edges (u, v) and (v, u) derived from (uv, t) inherit the same label t. But since such edges never appear in the same temporal path, their labels can be shifted-without affecting reachability-to restore a proper labeling. With this, we make the following observation:

▶ **Observation 5.1.** *It holds that* UD + x $\rightsquigarrow^{R}$ D + x*, but* D + x $\not\rightsquigarrow^{R}$ UD + x*.*

The remaining question is whether we can transform an undirected setting into a directed setting at "a level lower". For UD + strict and UD + strict + simple this is not the case.

▶ **Lemma 5.2** (UD + strict + simple $\not\rightsquigarrow^{R}$ D + non-strict)**.** *There exists a graph $\mathcal{G} \in$ UD + strict + simple such that there is no $\mathcal{H} \in$ D + non-strict with $\mathcal{R}(\mathcal{H}) = \mathcal{R}(\mathcal{G})$.*

Proof. Consider the following temporal graph $\mathcal{G}$ in the UD + strict + simple setting (left) and the corresponding reachability graph (right). For the sake of contradiction, let $\mathcal{H}$ be a temporal graph in the D + non-strict setting whose reachability graph is isomorphic to that



of $\mathcal{G}$. Note that in $\mathcal{G}$, a reaches both b and d, while d does not reach b. Therefore, the direct edge (a, b) must be included in $\mathcal{H}$ to ensure that a can reach b. By the same reasoning, every edge in $\mathcal{R}(\mathcal{G})$ has to be included in $\mathcal{H}$.

Consider the cycle a, b, c, d, a. Regardless of how the edges on this cycle are labeled, Lemma 4.1 implies that this will create at least one transitive reachability. For example, the labeling $(a, b, 4)$, $(b, c, 3)$, $(c, d, 2)$, $(d, a, 1)$, results in d reaching b. However, no such transitive reachability appears in $\mathcal{R}(\mathcal{G})$. Therefore, there exists no valid labeling for $\mathcal{H}$. ◀

Interestingly, among the non-strict and proper settings, there exists indeed one such "level-breaking" transformation. Applying the reachability-dilation (Definition 4.11) to UD + non-strict + simple graphs, produces a reachability equivalent D + proper + simple graph. Reachability dilation replaces each connected component in a snapshot with a bidirected spanning tree, preserving reachability while ensuring properness and a simple labeling. The proof follows from the proof for directed reachability dilation (Theorem 4.13). Refer to Figure 10 for an illustration of this transformation.

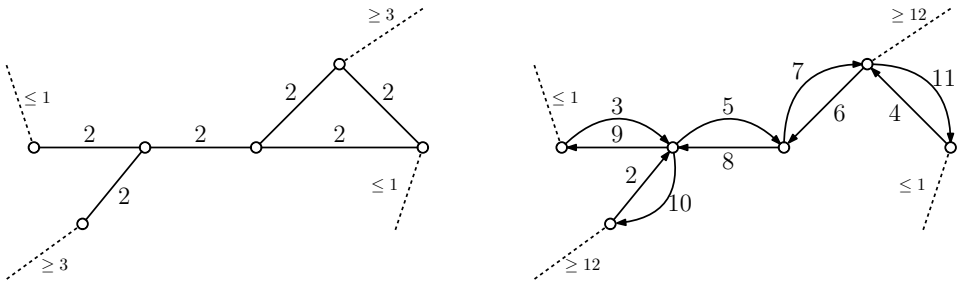


Figure 10 Illustration of the *reachability-dilation* (Definition 4.11) applied to an UD + non-strict + simple graph $\mathcal{G}$ with snapshot G_2 (left) and the proper labeling with the bidirected subtrees (right). The dotted edges represent adjacent earlier and later edges (shifted in the proper labeling).

▶ **Lemma 5.3** (UD + non-strict + simple $\rightsquigarrow^R$ D + proper + simple)**.** *Given a temporal graph* $\mathcal{G} \in$ UD + non-strict + simple*, reachability dilation transforms* $\mathcal{G}$ *into a a* D + proper + simple *temporal graph* $\mathcal{H}$ *with* $\mathcal{R}(\mathcal{G}) = \mathcal{R}(\mathcal{H})$*.*

Proof. ($\mathcal{H}$ is proper.) This follows from the same arguments as in Theorem 4.13.
($\mathcal{H}$ is simple.) Reachability-dilation replaces each connected component of a snapshot of $\mathcal{G}$ with a spanning tree of the underlying footprint. Since the component is connected, such a tree always exists. By this process, each edge uv within the component is either removed or replaced by two opposing directed temporal edges, each assigned one time label. As $\mathcal{G}$ is simple, no pair of vertices will be considered more than once, and consequently, in $\mathcal{H}$, each pair of vertices is connected by either zero or two opposing directed, single labeled edges.
($\mathcal{R}(\mathcal{G}) = \mathcal{R}(\mathcal{H})$.) This follows from the same arguments as in Theorem 4.13. ◀

6 Conclusion and Future Work

We extended the framework of Casteigts, Corsini, and Sarkar [8] from undirected to directed temporal graphs, analyzing how definitional choices of temporal graphs affect their reachability expressivity. Using the equivalence notions of support, reachability, and induced-reachability, we fully resolved the directed hierarchy, completed the undirected one, and nearly resolved their merged comparison.

Two natural questions remain: whether every undirected + non-strict graph can be transformed into a reachability-equivalent directed + proper + simple graph, or even into a directed + non-strict + simple graph.

Future work. A key structural direction is to characterize which static graphs can occur as reachability graphs in a given temporal setting. Our separating structures – such as the directed triangle – offer a first step toward such characterizations.

Another important direction is to study the efficiency of transformations between temporal settings. While this work focused on existence, future research could investigate how to minimize the number of time labels, preserve structural parameters such as treewidth or lifetime, or ensure compatibility with specific computational problems. Many (folklore) transformations already exist in the literature, and it could be valuable to collect, formalize, and analyze them within a unified framework.

Perhaps most promising is the potential impact on algorithmic research. Many hardness results and algorithms for temporal problems – such as shortest paths [23], connected components [4], exact edge covers [13], or temporal versions of Menger's Theorem [3] – are currently first shown in one setting and then reproven via ad hoc adaptations. Our framework opens the door to formalizing such generalizations by identifying which problems are invariant under equivalence notions or specific transformations. For example, maximum flow is preserved under the semaphore construction, and open connected components remain invariant under reachability equivalence. Establishing which problems admit such invariance would streamline future work, allowing results to transfer across settings without reproving them from scratch.

References

1 Eleni C. Akrida, George B. Mertzios, Paul G. Spirakis, and Leszek Gasieniec. The Complexity of Optimal Design of Temporally Connected Graphs. *Theory of Computing Systems / Mathematical Systems Theory*, 61(3):907–944, April 2017. Publisher: Springer US. doi:10.1007/S00224-017-9757-X.

2 Brenda Baker and Robert Shostak. Gossips and Telephones. *Discrete Mathematics*, 2(3):191–193, June 1972. doi:10.1016/0012-365X(72)90001-5.

3 Kenneth A. Berman. Vulnerability of scheduled networks and a generalization of Menger's Theorem. *Networks: An International Journal*, 28(3):125–134, 1996. doi:10.1002/(SICI)1097-0037(199610)28:3<125::AID-NET1>3.0.CO;2-P.

4 Sandeep Bhadra and Afonso Ferreira. Complexity of Connected Components in Evolving Graphs and the Computation of Multicast Trees in Dynamic Networks. In Samuel Pierre, Michel Barbeau, and Evangelos Kranakis, editors, *Ad-Hoc, Mobile, and Wireless Networks*, pages 259–270, Berlin, Heidelberg, 2003. Springer. doi:10.1007/978-3-540-39611-6_23.

5 Hans L. Bodlaender and Tom C. van der Zanden. On exploring always-connected temporal graphs of small pathwidth. *Information Processing Letters*, 142:68–71, February 2019. doi:10.1016/j.ipl.2018.10.016.

6 Richard T. Bumby. A Problem with Telephones. *SIAM Journal on Algebraic Discrete Methods*, 2(1):13–18, March 1981. Publisher: Society for Industrial and Applied Mathematics. doi:10.1137/0602002.

7 Arnaud Casteigts and Timothée Corsini. In search of the lost tree: Hardness and relaxation of spanning trees in temporal graphs, December 2023. arXiv:2312.06260 [cs]. doi:10.48550/arXiv.2312.06260.

8 Arnaud Casteigts, Timothée Corsini, and Writika Sarkar. Simple, Strict, Proper, Happy: A Study of Reachability in Temporal Graphs. *Theoretical Computer Science*, 991:114434, April 2024. doi:10.1016/j.tcs.2024.114434.

9 Arnaud Casteigts, Michael Raskin, Malte Renken, and Viktor Zamaraev. Sharp Thresholds in Random Simple Temporal Graphs. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 319–326, Denver, CO, USA, February 2022. IEEE. doi:10.1109/FOCS52979.2021.00040.

10 Esteban Christiann, Eric Sanlaville, and Jason Schoeters. On Inefficiently Connecting Temporal Networks. In Arnaud Casteigts and Fabian Kuhn, editors, *3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024)*, volume 292 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:19, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISSN: 1868-8969. doi:10.4230/LIPIcs.SAND.2024.8.

11 Isnard Lopes Costa, Raul Lopes, Andrea Marino, and Ana Silva. On Computing Large Temporal (Unilateral) Connected Components. In *Hsieh, SY., Hung, LJ., Lee, CW. (eds) Combinatorial Algorithms*, volume 13889 of *Lecture Notes in Computer Science*. Springer, February 2023. arXiv:2302.12068 [math]. doi:10.1007/978-3-031-34347-6_24.

12 Argyrios Deligkas, Michelle Döring, Eduard Eiben, Tiger-Lily Goldsmith, and George Skretas. Being an influencer is hard: The complexity of influence maximization in temporal graphs with a fixed source. *Information and Computation*, 299:105171, August 2024. doi:10.1016/j.ic.2024.105171.

13 Argyrios Deligkas, Michelle Döring, Eduard Eiben, Tiger-Lily Goldsmith, George Skretas, and Georg Tennigkeit. How Many Lines to Paint the City: Exact Edge-Cover in Temporal Graphs. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(25):26498–26506, April 2025. Number: 25. doi:10.1609/aaai.v39i25.34850.

14 Thomas Erlebach, Othon Michail, and Nils Morawietz. Recognizing and Realizing Temporal Reachability Graphs. In *33rd Annual European Symposium on Algorithms, ESA 2025*, LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPICS.ESA.2025.93.

15 Thomas Erlebach and Jakob T. Spooner. Exploration of k-edge-deficient temporal graphs. *Acta Informatica*, 59(4):387–407, August 2022. doi:10.1007/s00236-022-00421-5.

16 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. Publisher: Elsevier. doi:10.1016/J.TCS.2019.03.031.

17 Eugen Füchsle, Hendrik Molter, Rolf Niedermeier, and Malte Renken. Delay-Robust Routes in Temporal Graphs. In *39th International Symposium on Theoretical Aspects of Computer Science*, volume 219, pages 30:1–30:15, Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2022), 2022. Leibniz International Proceedings in Informatics (LIPIcs). arXiv:2201.05390 [cs]. doi:10.4230/LIPIcs.STACS.2022.30.

18 F. Göbel, J. Orestes Cerdeira, and H. J. Veldman. Label-Connected Graphs and the Gossip Problem. *Discrete Mathematics*, 87(1):29–40, January 1991. doi:10.1016/0012-365X(91)90068-D.

19 A. Hajnal, E. C. Milner, and E. Szemerédi. A Cure for the Telephone Disease. *Canadian Mathematical Bulletin*, 15(3):447–450, 1972. doi:10.4153/CMB-1972-081-0.

20 David Kempe, Jon Kleinberg, and Amit Kumar. Connectivity and Inference Problems for Temporal Networks. *Journal of Computer and System Sciences*, 64(4):820–842, June 2002. doi:10.1006/jcss.2002.1829.

21 Nina Klobas, George B. Mertzios, Hendrik Molter, and Paul G. Spirakis. Temporal Graph Realization from Fastest Paths. *LIPIcs, Volume 292, SAND 2024*, 292:16:1–16:18, 2024. Publisher: Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPICS.SAND.2024.16.

22 Mathilde Vernet, Maciej Drozdowski, Yoann Pigné, and Eric Sanlaville. A Theoretical and Experimental Study of a New Algorithm for Minimum Cost Flow in Dynamic Graphs. *Discrete Applied Mathematics*, 296:203–216, June 2021. doi:10.1016/j.dam.2019.12.012.

23 B. Xuan, Afonso Ferreira, and Aubin Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks, October 2002.

Pathfinding in Self-Deleting Graphs

Michal Dvořák
Czech Technical University, Prague, Czech Republic

Dušan Knop
Czech Technical University, Prague, Czech Republic

Michal Opler
Czech Technical University, Prague, Czech Republic

Jan Pokorný
Czech Technical University, Prague, Czech Republic

Ondřej Suchý
Czech Technical University, Prague, Czech Republic

Krisztina Szilágyi
Czech Technical University, Prague, Czech Republic

Abstract

In this paper, we study the problem of pathfinding on traversal-dependent graphs, i.e., graphs whose edges change depending on the previously visited vertices. In particular, we study *self-deleting graphs*, introduced by Carmesin et al. [5], which consist of a graph $G = (V, E)$ and a function $f\colon V \to 2^E$, where $f(v)$ is the set of edges that will be deleted after visiting the vertex v. In the (Shortest) Self-Deleting s-t-path problem we are given a self-deleting graph and its vertices s and t, and we are asked to find a (shortest) path from s to t, such that it does not traverse an edge in $f(v)$ after visiting v for any vertex v.

We prove that Self-Deleting s-t-path is NP-hard even if the given graph is outerplanar, bipartite, has maximum degree 3, bandwidth 2 and $|f(v)| \leq 1$ for each vertex v. We show that Shortest Self-Deleting s-t-path is W[1]-complete parameterized by the length of the sought path and that Self-Deleting s-t-path is W[1]-complete parameterized by the vertex cover number, feedback vertex set number and treedepth. We also show that the problem becomes FPT when we parameterize by the maximum size of $f(v)$ and several structural parameters. Lastly, we show that the problem does not admit a polynomial kernel even for parameterization by the vertex cover number and the maximum size of $f(v)$ combined already on 2-outerplanar graphs.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Parameterized complexity, self-deleting graphs, pathfinding

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.28

Related Version *Full Version*: https://arxiv.org/abs/2507.12047

Funding This work was co-funded by the European Union under the project Robotics and advanced industrial production (reg. no. CZ.02.01.01/00/22_008/0004590). Michal Dvořák and Jan Pokorný acknowledge the support of Grant Agency of the Czech Technical University in Prague, grant No. SGS23/205/OHK3/3T/18.

1 Introduction

Pathfinding in graphs is a well-studied topic, both from a theoretical and from a practical perspective. The famous Dijkstra's algorithm for finding a shortest path between two vertices runs in time that is quadratic in the number of vertices. However, this algorithm works under the assumption that the underlying graph is static, i.e., does not change. In many practical applications, this assumption does not hold.

© Michal Dvořák, Dušan Knop, Michal Opler, Jan Pokorný, Ondřej Suchý, and Krisztina Szilágyi; licensed under Creative Commons License CC-BY 4.0

One way to reflect the changes in the underlying graph is to model it as a temporal graph, where each edge is present in the graph only at certain times. In this setting, there are several ways to define the "best" path between two vertices: shortest path (using the smallest number of edges), fastest (the path that requires the smallest number of timesteps) and foremost (the path that requires the smallest number of timesteps starting from time zero). In all of these cases, the pathfinding problem can be solved in polynomial time [23].

In some applications, however, the way that the underlying graph changes is *traversal-dependent*. In other words, the availability of an edge is not time-dependent, but it depends on the previously visited vertices and edges. One such example is open-pit mining, where drilling at a vertex creates a pile of rubble which renders some edges impassable. Another example is autonomous harvesting, in which the vehicle should not return to the previously visited areas to avoid soil compactification [5].

In this paper, we consider the model introduced by Carmesin et al. [5], called *self-deleting graph*. A self-deleting graph is a graph $G = (V, E)$ together with a function $f\colon V \to 2^E$, which describes which edges will be deleted after visiting a vertex. We stress that the edges in $f(v)$ are not necessarily incident to v. We consider the (Shortest) Self-Deleting s-t path problem, where we are given a self-deleting graph and its vertices s and t and we are asked to find a (shortest) path from s to t that is valid (i.e., in which we are not traversing an edge in $f(v)$ after visiting the vertex v).

Related work. Several variants of pathfinding with restrictions on vertices and edges have been studied.

Wojciechowski et al. [21] introduced a problem called Optional Choice Reachability, where we are given a graph together with a set S of pairs of edges, and we are asked to find an s-t path that contains at most one edge from each pair in S. They showed that this problem is NP-complete even on directed acyclic graphs (DAGs) of pathwidth 2 and FPT parameterized by $|S|$.

The vertex analogue of this problem, where we are given a set of pairs of vertices and we are required to use at most one of them from each pair was introduced by Krause et al. [19]. On the one hand, the problem is NP-hard even for DAGs [13], even if the set of pairs has a specific structure, such as overlapping [17] or ordered [18]. On the other hand, it is polynomial time solvable, if the structure is well-parenthesized [17], halving [17], or nested [9], or in other special cases [24]. Notably, Bodlaender et al. [3] studied the problem from parameterized perspective on undirected graphs, showing that it is W[1]-hard w.r.t. vertex cover number of the input graph G, but FPT w.r.t. vertex cover number of graph H or treewidth of graph $G \cup H$, where graph H has an edge for each forbidden pair. Moreover, it does not admit polynomial kernel w.r.t. vertex cover number of graph $G \cup H$, unless $\mathsf{NP} \subseteq \mathsf{coNP}/_{\mathrm{poly}}$ [3].

Szeider [20] studied the problem of finding a path where for each vertex, only certain combinations of incoming and outgoing edges are permitted. He provides a dichotomy between the cases which are NP-hard and cases which are linear time solvable, based on the structure of permitted combinations. In the variant where the cost of each arc depends on previously traversed arcs, introduced by Kirby and Potts [16], the shortest walk can be found in polynomial time [2, 25], but finding the shortest path is NP-hard and hard to approximate [22]. In the variant with exclusive-disjunction arc pairs conflicts, introduced by Cerulli et al. [6], we are given pairs of arcs and we have to pay a penalty if the path contains either none of the arcs or both of them. The goal is to find a path minimizing the sum of length of edges and the penalties paid. They show that the problem is NP-hard and provide heuristics.

Our results. In Section 3, we consider the (classical) complexity of (Shortest) Self-Deleting s-t-path. Our first result is that Self-Deleting s-t-path is NP-hard even on a very restricted graph class, namely outerplanar bipartite graphs of maximum degree 3. Moreover, as we show in Corollary 3.4, the result holds even when the self-deleting graph deletes at most one edge for each vertex (i.e., $|f(v)| \leq 1$ for all v). Next, we show a separation between Self-Deleting s-t-path and Shortest Self-Deleting s-t-path: namely, on cactus graphs the former problem can be solved in linear time, whereas the latter is NP-hard (Theorems 3.10 and 3.11 respectively).

In Section 4, we turn our attention to parameterized complexity. Firstly, we consider the parameterization of Shortest Self-Deleting s-t-path by solution size, and in Theorem 4.3 we show that it is W[1]-complete. For most structural parameters, Self-Deleting s-t-path turns out to be para-NP-hard or W[1]-complete. For an overview of our results on parameterizations by structural parameters, see Figure 1.

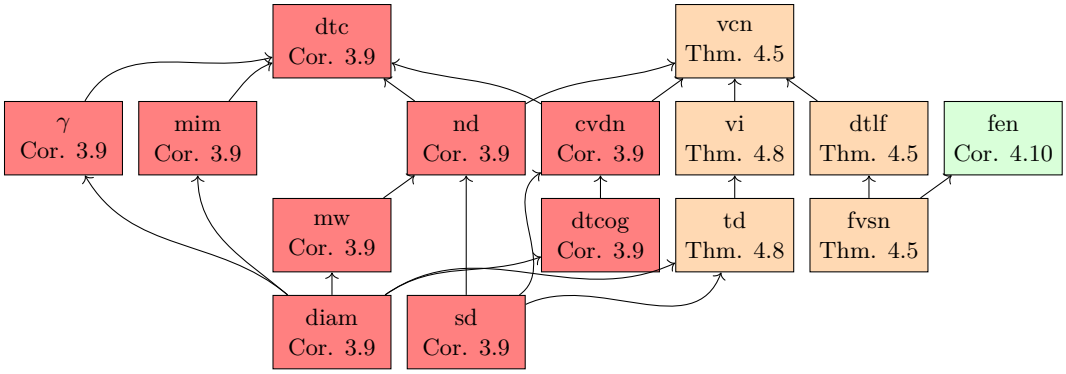


Figure 1 Results for Self-Deleting s-t-path parameterized by structural parameters. Red color stands for para-NP-hard, green is FPT and orange is W[1]-complete. Each parameter is accompanied by the corresponding statement from which the result follows. An arrow $\alpha \to \beta$ indicates a functional upper bound, i.e., $\alpha \leq g(\beta)$ for some (computable) function g. For full names of the parameters refer to Section 2.

In order to make the problem tractable, we consider the case of bounded deletion set size (i.e. we parameterize the problem by $\mu = \max_{v \in V} |f(v)|$). Although this parameterization alone does not make the problem tractable (in particular, we show that Shortest Self-Deleting s-t-path is para-NP-hard parameterized by μ), it turns out that the parameterization by μ and k (the number of vertices of the sought path) leads to an FPT algorithm. This result allows us to obtain several FPT algorithms for parameterizations by μ and structural parameters, such as vertex cover number, vertex integrity and treedepth. Using the results of Dvořák et al. [12] on the number of edges in traceable graphs with dense structure, we obtain FPT algorithms for several dense parameters and μ, such as neighborhood diversity, shrub-depth, modular-width, and size of maximum induced matching. For an overview of our results about parameterizations by structural parameters and μ, refer to Figure 2.

Lastly, in Section 5, we consider kernelizing the problem. We show that, under standard parameterized complexity assumptions, Self-Deleting s-t-path does not admit a polynomial kernel parameterized by μ and vertex cover number even on 2-outerplanar graphs. On the positive side, we obtain a linear kernel for the parameterization by feedback edge number and linear kernel for the parameterization by vertex cover number on outerplanar graphs. Moreover, while Self-Deleting s-t-path does not admit a classical kernel parameterized by μ on cliques, we obtain a linear Turing kernel on cliques parameterized by μ.

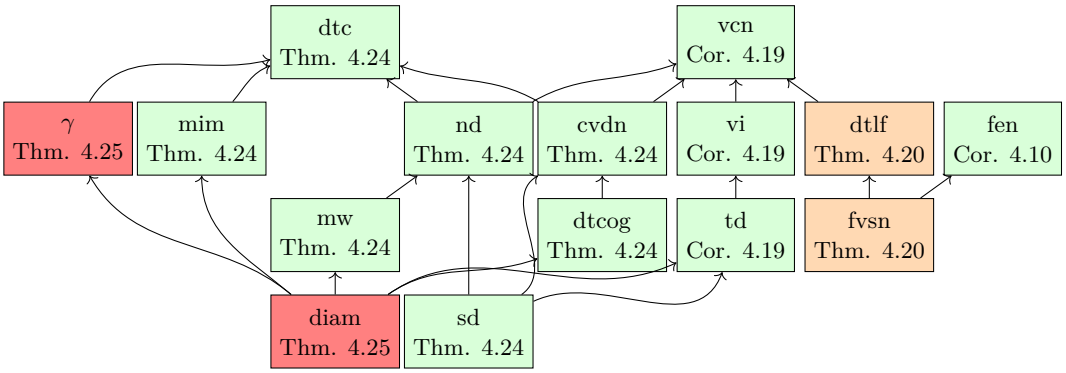


Figure 2 Results for Self-Deleting s-t-path parameterized by structural parameters and μ combined. The meaning of colors and arrows is the same as in Figure 1.

2 Preliminaries

For an integer k we let $[k] = \{1, 2, \ldots, k\}$. For any function $f\colon A \to B$ and $X \subseteq A$, the restriction of f to X is denoted $f|_X$. We use the notation $\widetilde{O}(\cdot)$ to suppress polylogarithmic factors in time complexity.

Graph Theory. We use standard notations, terminology, and definitions of graph theory, refer to Diestel [11] for those. Unless explicitly stated, we consider simple undirected graphs. We denote a v_1–v_k walk (or path) in a graph by the sequence $(v_1, e_1, v_2, \ldots, e_{k-1}, v_k)$ of vertices v_i and edges e_i, where $e_i = \{v_i, v_{i+1}\}$. For brevity, we may sometimes omit the edges and write just the sequence of vertices. We denote by $\mathrm{cc}(G)$ the number of connected components of a graph G. A graph is a *cactus* if every edge lies on at most one cycle. A graph is a *block graph* if every vertex-2-connected component is a clique. A $2 \times \ell$ grid for some positive integer ℓ is a *ladder*. The class of *cographs* is the minimal class of graphs containing the one-vertex graph and closed under complete joins and disjoint unions of two graphs.

Self-deleting graphs. A *self-deleting graph* is an ordered pair (G, f), where $G = (V, E)$ is an undirected graph and f is a function $f\colon V \to 2^E$. We write G instead of (G, f) if f is clear from the context. The function f is called the *deletion function*. Let (G, f) be a self-deleting graph. A path or walk $(v_1, e_1, v_2, e_2, \ldots, e_{k-1}, v_k)$ in G is *f-conforming* if for every $j \leq i$ we have $e_i \notin f(v_j)$. We denote $\mu_f = \max_{v \in V} |f(v)|$ and $|f| = \sum_{v \in V} |f(v)|$. If f is clear from the context, we omit the lower index and write just μ. Central to our paper are the two decision problems Self-Deleting s-t-path and Shortest Self-Deleting s-t-path. In Self-Deleting s-t-path we are given a self-deleting graph (G, f) and two vertices $s, t \in V(G)$ and the task is to decide if there is an f-conforming s-t path in G. In Shortest Self-Deleting s-t-path we are in addition given a positive integer k and the task is to decide if there is an f-conforming s-t path in G on at most k vertices.

We assume that $\mu \geq 1$ as otherwise the problems are trivial. Moreover, if μ appears in the base of an exponential or in a logarithm, μ should be replaced by $\max\{\mu, 2\}$ in order to avoid degenerate cases. We stick to writing μ for the sake of readability.

Parameterized complexity. We assume the reader is familiar with parameterized complexity. For definitions and comprehensive overview, refer to the monograph of Cygan et al. [10]. In our work we consider the following structural parameters: *bandwidth* (bw), *distance to*

clique (dtc), *vertex cover number* (vcn), *domination number* (γ), *maximum induced matching* (mim), *neighborhood diversity* (nd), *diameter* (diam), *cluster vertex deletion number* (cvdn), *feedback vertex set number* (fvsn), *distance to linear forest* (dtlf), *vertex integrity* (vi), *feedback edge number* (fen), *treedepth* (td), *distance to cograph* (dtcog), *modular-width* (mw), and *shrub-depth* (sd). Formal definitions of all the parameters can be found in the full version of this paper.

Exponential Time Hypothesis. Exponential Time Hypothesis (ETH), introduced by Impagliazzo, Paturi and Zane [14, 15] asserts, roughly speaking, that there is no algorithm for 3Sat in time $2^{o(n')}$, where n' is the number of variables of the input formula. In fact, even $2^{o(n'+m')}$ algorithm is ruled out by using the Sparsification Lemma [15], where m' is the number of clauses of the input formula. We also utilize the result of Chen, Huang, Kanj, and Xia [7, 8] that there is no $g(k)n^{o(k)}$ algorithm for (Multicolored) Clique for any computable function g unless ETH fails.

Statements where proofs or details are omitted due to space constraints are marked with $\star$. The omitted material is available in the full version of this paper.

3 Classical Complexity

As observed by Carmesin et al. [5, Lemma 3], if every vertex deletes only its incident edges, Self-Deleting s-t-path reduces to pathfinding in a directed graph: deleting $\{u, v\}$ at v orients the edge from u to v, and if both endpoints delete it, the edge is removed entirely. Thus, Self-Deleting s-t-path is solvable in linear time when all $f(v)$ contain only edges incident to v. Another scenario when the problem is tractable is that the graph has only a constant number of s-t paths, e.g., in trees or graphs of maximum degree 2. The problem becomes hard already on graphs of maximum degree 3. We show a polynomial-time reduction from 3Sat to Self-Deleting s-t-path which we also utilize later with slight modifications.

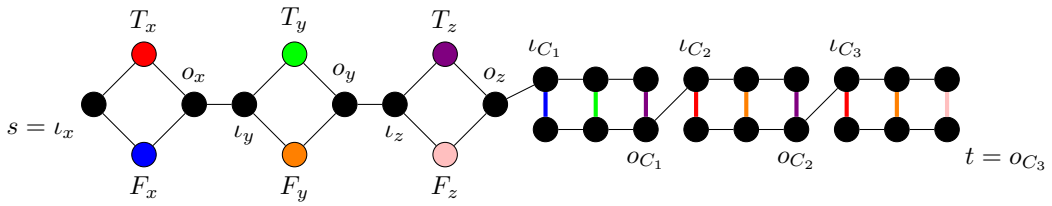


Figure 3 Example of Construction 3.1 for the input formula $\varphi = (x \vee \neg y \vee \neg z) \wedge (\neg x \vee y \vee \neg z) \wedge (\neg x \vee y \vee z)$. The deletion sets are indicated by the colors.

▶ **Construction 3.1.** Let φ be a given CNF-formula over variables $X = \{x_1, \ldots, x_{n'}\}$ with clauses $\mathcal{C} = \{C_1, \ldots C_{m'}\}$. Our construction of an instance of Self-Deleting s-t-path consist of variable and clause gadgets that we now introduce. See Figure 3 for an example.

Variable gadget The variable gadget for a variable $x \in X$ is a 4-cycle with vertices T_x, ι_x, F_x, o_x.

Clause gadget The clause gadget for a clause $C \in \mathcal{C}$ is the $2 \times |C|$ grid. The top left corner vertex is denoted ι_C (input) and the bottom right corner vertex is denoted o_C (output). Each of the $|C|$ vertical edges correspond to a literal of C. For a literal $\ell \in C$ we denote its corresponding edge by e^C_ℓ.

Create one variable gadget for each variable and one clause gadget for each clause. Add edges of the form $\{o_{x_k}, \iota_{x_{k+1}}\}$ for every $k \in [n'-1]$, add edge $\{o_{x_{n'}}, \iota_{C_1}\}$ and finally the edges $\{o_{C_i}, \iota_{C_{i+1}}\}$ for every $i \in [m'-1]$. We let $s = \iota_{x_1}$ and $t = o_{C_{m'}}$. It remains to specify the deletion function f. For all vertices v except T_x, F_x inside the variable gadgets, we set $f(v) = \emptyset$. For a variable $x \in X$ we set $f(T_x) = \{e^C_{\neg x} \mid C \in \mathcal{C}, \neg x \in C\}$ and $f(F_x) = \{e^C_x \mid C \in \mathcal{C}, x \in C\}$. In other words, T_x deletes the literal edges in clause gadgets corresponding to literals $\neg x$ and F_x deletes the literal edges corresponding to literals x. The resulting Self-Deleting s-t-path instance is (G, f, s, t).

▶ **Lemma 3.2** (⋆). *Let φ be the formula and (G, f, s, t) the Self-Deleting s-t-path instance obtained by Construction 3.1 from φ. Then φ is satisfiable if and only if there is an f-conforming s-t path in G.*

By using Construction 3.1 together with Lemma 3.2 we immediately obtain the following theorem.

▶ **Theorem 3.3.** *Self-Deleting s-t-path is* NP*-hard.*

▶ **Corollary 3.4.** *Self-Deleting s-t-path is* NP*-hard even if the deletion function f satisfies $\forall v \in V : |f(v)| \leq 1$, i.e., $\mu \leq 1$.*

Proof. Use Construction 3.1 with the following modification. Replace the vertex T_x (resp. F_x) by a path on $|f(T_x)|$ (resp. $|f(F_x)|$) vertices and delete one edge of the original set $f(v)$ on each vertex of the path to obtain $|f(v)| \leq 1$. ◀

▶ Remark 3.5. Note that Construction 3.1 produces a graph with $O(n' + m')$ vertices and edges, where n' and m' are the number of variables and clauses in the original 3Sat formula.

▶ **Lemma 3.6** (⋆). *$2 \times \ell$ grid has bandwidth 2, treedepth at most $O(\log \ell)$ and vertex integrity at most $O(\sqrt{\ell})$.*

Since the resulting graph of Construction 3.1 is a subgraph of a ladder of size $O(n' + m')$, it is straightforward to obtain the following corollary:

▶ **Corollary 3.7.** *Self-Deleting s-t-path is* NP*-hard even when restricted to outerplanar bipartite graphs of maximum degree 3 and bandwidth 2 even with $\mu \leq 1$.*

▶ **Corollary 3.8** (⋆). *Self-Deleting s-t-path remains* NP*-hard even when restricted to the classes of unit interval graphs, ladder graphs, or block graphs even when $\mu \leq 1$.*

We can also easily obtain hardness on cliques by adding all non-existent incident edges to the deletion set of every vertex. However, as we will see later, we cannot hope to upper bound μ by a constant in cliques and preserve NP-hardness (Observation 4.21 and Corollary 5.3).

▶ **Corollary 3.9** (⋆). *Self-Deleting s-t-path is* NP*-hard even when restricted to cliques.*

Cactus graphs. Corollaries 3.7 and 3.8 show that even if we allow a slight generalization of trees, the problem becomes NP-hard. One particular non-trivial case that allows for a polynomial-time algorithm is the class of cactus graphs. We show that when restricted to cactus graphs, Self-Deleting s-t-path becomes polynomial-time solvable (Theorem 3.10). Interestingly, deciding an existence of an f-conforming path is easy, however Shortest Self-Deleting s-t-path remains NP-hard even in cactus graphs (Theorem 3.11). Note that in cactus graphs each edge lies on at most 1 cycle, whereas the graph resulting from Construction 3.1 has each edge on at most 2 cycles so the problem becomes NP-hard in this case.

▶ **Theorem 3.10** (⋆). *Self-Deleting s-t-path can be solved in linear time if the underlying graph is a cactus.*

Proof Sketch. Consider the block-cut tree of G and note that we can only restrict ourselves to the part of G that contains the s-t path in the block-cut tree of G. This part consists of cycles and bridges. In each cycle we have two options to choose from. Each choice possibly forbids some choices in the future. This can be encoded as an implication of the type *if we choose this path, then we cannot choose some other path in the future.* We can thus reduce the problem to 2Sat. Details can be found in the full version of this paper. ◀

▶ **Theorem 3.11** (⋆). *Shortest Self-Deleting s-t-path is* NP*-hard even when restricted to cactus graphs, and* $\mu \leq 1$.

Proof Sketch. We provide a polynomial reduction from Independent Set. For each vertex we introduce one cycle on the designated s-t-path (in the block-cut tree). One of the paths corresponds to taking the vertex and the other to not taking it. Inclusion of a vertex forbids inclusion of any of its neighbors by deleting an edge of the appropriate path. Crucially, the inclusion path is shorter, so the desired length of the path forces an appropriate number of vertices to be included in the independent set. ◀

4 Parameterized complexity

In this section we focus on parameterized complexity of (Shortest) Self-Deleting s-t-path. Note that the problem is para-NP-hard parameterized by bandwidth or maximum degree (hence also parameterized by treewidth) due to Corollary 3.7. Moreover, it is also para-NP-hard parameterized by any parameter that is constant on cliques (Corollary 3.9).

We show that Shortest Self-Deleting s-t-path is W[1]-complete parameterized by k (the number of vertices of the sought path) (Theorem 4.3) and that Self-Deleting s-t-path is W[1]-hard parameterized by the vertex cover number. We then show that Self-Deleting s-t-path is in fact W[1]-complete for parameters vertex cover number, vertex integrity, treedepth, distance to linear forest, and feedback vertex set number (Theorems 4.5 and 4.8).

The last hope for positive algorithmic results lies in the parameter feedback edge number (fen). Here, we observe that Self-Deleting s-t-path parameterized by fen can be solved in $O(2^{\text{fen}}(n+m+|f|))$ time (Corollary 4.10). Later, in Section 5 we show that Self-Deleting s-t-path even admits kernel with $O(\text{fen})$ vertices and edges (however, this does not yield as fast algorithm). Refer to Figure 1 for a graphical overview of the results for structural parameters alone.

Then, in order to obtain algorithmic results, we combine structural parameters with the parameter $\mu = \max_{v \in V} |f(v)|$. While Shortest Self-Deleting s-t-path is para-NP-hard parameterized by μ alone (Corollary 3.4), and W[1]-complete parameterized by k (Theorem 4.3), the problem becomes FPT parameterized by k and μ combined (Theorem 4.17). This also yields many FPT results for Self-Deleting s-t-path parameterized by structural parameters and μ combined (Theorem 4.24). Refer to Figure 2 for an overview of such results.

4.1 W[1]-completeness for length of the path

We begin with a reduction from Multicolored Clique to Self-Deleting s-t-path. We developed this construction independently, although later we discovered that it is similar to the one of Bodlaender et al. [3, Theorem 9] for a related problem.

▶ **Construction 4.1.** Let $G = (V, E)$ be the input graph for Multicolored Clique and let $V = V_1 \cup \cdots \cup V_k$ be the partition of V into k color classes. We build an instance (G', f, s, t) of Self-Deleting s-t-path as follows. Create $k + 1$ guard vertices $g_0, g_1, \ldots, g_k$. For every $i \in [k]$ and $v \in V_i$ add a vertex y_v and connect it to g_{i-1} and g_i by edges $e_1^v = \{g_{i-1}, y_v\}$ and $e_2^v = \{y_v, g_i\}$. We denote by P_v^i the path $(g_{i-1}, e_1^v, y_v, e_2^v, g_i)$. This completes the description of the graph G'. We let $s = g_0$ and $t = g_k$. It remains to specify the deletion sets. The only vertices with nonempty deletion sets will be the vertices y_v on the paths P_v^i. We let $f(y_v) = \bigcup\{e_1^w, e_2^w \mid w \in V_j, j \neq i, \{w, v\} \notin E(G)\}$.

▶ **Lemma 4.2** (⋆). *Let $(G, V_1, \ldots, V_k)$ be an instance of Multicolored Clique and (G', f, s, t) be the instance of Self-Deleting s-t-path obtained from it by Construction 4.1. There is a multicolored clique in G if and only if there is an f-conforming s-t path in G'.*

Note that in the instance (G', f, s, t) from Construction 4.1, any f-conforming s-t path has at most $2k+1$ vertices. We immediately obtain W[1]-hardness of Shortest Self-Deleting s-t-path w.r.t. k.

▶ **Theorem 4.3** (⋆). *Shortest Self-Deleting s-t-path is* W[1]*-complete w.r.t. k.*

4.2 Parameterization by structural parameters alone

Observe that the resulting graph G' from Construction 4.1 has vertex cover number at most $k + 1$, because $G' \setminus \{g_0, \ldots, g_k\}$ is edgeless. We immediately obtain W[1]-hardness of Self-Deleting s-t-path for the parameter vertex cover number.

We establish membership in W[1] of Self-Deleting s-t-path parameterized by the feedback vertex set number (Theorem 4.4) and treedepth (Observation 4.7). These results together with W[1]-hardness for vertex cover number establish W[1]-completeness for the following parameters: vertex cover, distance to linear forest, feedback vertex set, vertex integrity, and treedepth (Theorems 4.5 and 4.8).

▶ **Theorem 4.4** (⋆). *Self-Deleting s-t-path parameterized by the feedback vertex set number is in* W[1].

Proof Sketch. If G has a feedback vertex set S, then any path in G is split by S into at most $|S| + 1$ segments where the segments are uniquely determined by their endpoints, as $G \setminus S$ is a forest. We can thus equivalently look for a path of length $O(\text{fvsn})$ in a graph where we represent long paths in $G \setminus S$ by paths of length two and reflect the deletions along these unique u-v paths in $G \setminus S$. Full proof can be found in the full version of this paper. ◀

▶ **Theorem 4.5** (⋆). *Self-Deleting s-t-path parameterized by feedback vertex set, distance to linear forest, or vertex cover is* W[1]*-complete, solvable in $n^{O(\alpha)}$ time and unless ETH fails, there is no $g(\alpha)n^{o(\alpha)}$ algorithm for any of the above parameters and any computable function g.*

▶ **Lemma 4.6** (⋆). *Let G be a graph with treedepth* td *and vertex integrity* vi*. Then G contains no path on more than 2^{td} or $\text{vi}^2 + 2\,\text{vi}$ vertices.*

▶ **Observation 4.7.** *Self-Deleting s-t-path parameterized by treedepth is in* W[1].

Proof. We provide a parameterized reduction from Self-Deleting s-t-path parameterized by treedepth to Self-Deleting s-t-path parameterized by k, which is in W[1] by Theorem 4.3. Let (G, f, s, t) be an instance of Self-Deleting s-t-path, we return the instance (G, f, s, t, k) where $k = 2^{\text{td}(G)}$. Correctness follows from Lemma 4.6. ◀

▶ **Theorem 4.8** (⋆). *Self-Deleting s-t-path is* W[1]*-complete parameterized by treedepth or vertex integrity. More precisely, Self-Deleting s-t-path can be solved in* $n^{O(2^{\mathrm{td}})}$ *and* $n^{O(\mathrm{vi}^2)}$ *time. For any* $\varepsilon > 0$*, algorithms for Self-Deleting s-t-path with running times* $n^{O(\mathrm{vi}^{2-\varepsilon})}\operatorname{poly}(n)$ *or* $n^{2^{o(\mathrm{td})}}\operatorname{poly}(n)$ *violate ETH.*

We complement the hardness results by a positive result for the parameter feedback edge number. Note that given a path P in a self-deleting graph, it is easy to check whether P is f-conforming in time $O(n+m+|f|)$.

▶ **Lemma 4.9** (⋆). *Let G be a graph and $s,t \in V(G)$ two fixed vertices in G. Then the number of s-t-paths in G is at most $2^{\mathrm{fen}(G)}$.*

▶ **Corollary 4.10** (⋆). *Self-Deleting s-t-path can be solved in $O(2^{\mathrm{fen}(G)}(n+m+|f|))$ time. Moreover, $2^{o(\mathrm{fen})}\operatorname{poly}(n)$-time algorithm for Self-Deleting s-t-path violates ETH.*

4.3 FPT algorithm for k and μ combined

We demonstrate that Shortest Self-Deleting s-t-path parameterized by k and μ combined is in FPT. We utilize color-coding, introduced by Alon et al. [1]. We consider a colorful variant of the problem, where we consider coloring of the edges and we only distinguish edges based on their colors. The sought solution – an f-conforming path $P = (v_1, e_1, \ldots, e_{r-1}, v_r)$ on $r \leq k$ vertices interacts with the edges e_i in the path and with the edges in the deletion sets $f(v_1), \ldots, f(v_r)$. By assumption there are at most $\mu k + k - 1$ such edges in total. Hence if we can ensure that the coloring will behave nicely on the set $\{e_1, \ldots, e_{r-1}\} \cup \bigcup_{i=1}^{r} f(v_i)$, we will find the path even in the colorful variant. We now formally define the colorful variant of our problem, which we refer to as Shortest χ-compliant s-t-path.

▶ **Definition 4.11.** *Let (G,f) be a self-deleting graph and let $\chi\colon E(G) \to [q]$ be a coloring of its edges. Let $P = (v_1, e_1, \ldots, e_{r-1}, v_r)$ be a path in G. We say that*

1. *P is* χ-compliant *if $\chi(e_i) \notin \chi(f(v_j))$ for any $j \leq i$,*
2. *P is* half-χ-rainbow *if $\chi(\bigcup_{i=1}^{r} f(v_i) \setminus E(P)) \cap \chi(E(P)) = \emptyset$ and $\chi|_{E(P)}$ is injective.*
3. *P is* χ-rainbow *if for $F = E(P) \cup \bigcup_{i=1}^{r} f(v_i)$ the restriction $\chi|_F$ is injective.*

In the Shortest χ-compliant s-t-path problem we are given a self-deleting graph (G,f), positive integer k, coloring $\chi\colon E(G) \to [q]$, vertices $s,t \in V(G)$ and the task is to decide whether there is a χ-compliant s-t path on at most k vertices in G.

▶ **Lemma 4.12** (⋆). *The following holds for any path P:*

1. *P is χ-rainbow $\Rightarrow$ P is half-χ-rainbow.*
2. *P is half-χ-rainbow and f-conforming $\Rightarrow$ P is χ-compliant.*
3. *P is χ-compliant $\Rightarrow$ P is f-conforming.*

▶ **Lemma 4.13** (⋆). *Given an instance of Shortest χ-compliant s-t-path and a set of colors $Q \subseteq [q]$, we can in $O(2^{|Q|}(n+m+|f|))$ time decide whether there exists a χ-compliant path on at most k vertices using only colors from Q.*

▶ **Lemma 4.14.** *Shortest χ-compliant s-t-path can be solved in $O(2^q(n+m+|f|))$ or in $O(\binom{q}{k}2^k(n+m+|f|))$ time.*

Proof. We can either plug $Q = [q]$ into the algorithm of Lemma 4.13 and obtain the running time $O(2^q(n + m + |f|))$ or we can try all possible sets Q of $k - 1$ colors and obtain the running time $O(\binom{q}{k}2^k(n+m+|f|))$. Note that we seek a path on (at most) k vertices, hence (at most) $k - 1$ edges. ◀

▶ **Theorem 4.15** (⋆). *There is a randomized algorithm solving* Shortest Self-Deleting *s-t-path with the following guarantees. Given $\varepsilon \in (0, 1)$, it runs in $2^{O(k \log \mu)}(n + m) \ln \frac{1}{\varepsilon}$ time. Moreover, if the input is a no-instance, the algorithm outputs no. If the input is a yes-instance, the algorithm outputs yes with probability at least $1 - \varepsilon$.*

Proof sketch. Asume that $\mu \geq 1$ as otherwise the problem is trivial. We use the algorithm from Lemma 4.14 with running time $O(\binom{q}{k}2^k(n + m + |f|))$. We try a random edge coloring χ using $q = 4k\mu$ colors. For a fixed coloring using q colors, we lower bound the probability that a path P is χ-compliant, given that P is f-conforming. Thus, we are equivalently lower bounding the probability that the algorithm succeeds in finding the f-conforming path. Let $P = (v_1, e_1, \ldots, e_{r-1}, v_r)$. We lower bound the probability that P is half-χ-rainbow. Suppose that the edges in $\bigcup_{i=1}^{r} f(v_i)$ are colored by $w \leq |\bigcup_{i=1}^{r} f(v_i) \setminus E(P)| \leq r\mu \leq k\mu$ colors from the set $[q]$. For P to become half-χ-rainbow, we need to ensure that the edges in $E(P)$ are colored by one of the remaining $q - w \geq 4k\mu - k\mu = 3k\mu$ colors and that the coloring is injective on $E(P)$. The probability of that happening for the $r - 1$ edges is $\frac{q-w}{q} \cdot \frac{q-w-1}{q} \cdot \ldots \cdot \frac{q-w-r+2}{q}$. Note that the last term lower bounds every other term and the last term is lower bounded by $\frac{1}{2}$ because $\frac{q-w-r+2}{q} \geq \frac{3k\mu-k}{4k\mu} \geq \frac{1}{2}$ given that $\mu \geq 1$. Hence the probability that P is half-χ-rainbow is at least $2^{-r+1} \geq 2^{-k}$.

We repeat the above for $2^k \ln \frac{1}{\varepsilon}$ random choices of χ and the proof on the probability of error and running time follow. The full proof can be found in the full version of this paper. ◀

There are two ways to derandomize the algorithm given in Theorem 4.15. Neither of the two algorithms is Pareto optimal with respect to μ and k. Hence, each of them turns out to be more suitable in different scenarios.

▶ **Theorem 4.16** (⋆). *Shortest Self-Deleting s-t-path is solvable in $2^{O(\mu k)}(n+m) \log n$ time.*

▶ **Theorem 4.17** (⋆). *Shortest Self-Deleting s-t-path is solvable in $2^{O(k \log(k\mu))}(n + m) \log n$ time.*

4.4 Parameterization by structural parameters and μ combined

We utilize the FPT algorithms w.r.t. k and μ combined to obtain several FPT algorithms for various structural parameters combined with μ. In the results that follow, we apply whichever of Theorem 4.17 or Theorem 4.16 yields the better asymptotic running time for the given parameters.

We start with the vertex cover number for which we prove similar bound as in Lemma 4.6.

▶ **Observation 4.18** (⋆). *Let G be a graph with vertex cover number* vcn*. Then G contains no path on more than $2\,\mathrm{vcn} + 1$ vertices.*

▶ **Corollary 4.19** (⋆). *Self-Deleting s-t-path can be solved in* $2^{O(\mu \cdot \mathrm{vcn})}(n+m)\log n$ *time, in* $2^{O(\mu \cdot \mathrm{vi}^2)}(n+m)\log n$ *time, or in* $2^{O(\mu \cdot 2^{\mathrm{td}})}(n+m)\log n$ *time. Moreover, algorithms with running times* $2^{o(\mathrm{vcn})}\,\mathrm{poly}(n)$, $2^{o(\mathrm{vi}^2)}\,\mathrm{poly}(n)$, *or* $2^{2^{o(\mathrm{td})}}\,\mathrm{poly}(n)$ *for Self-Deleting s-t-path even for* $\mu = 1$ *violate ETH.*

Unless FPT = W[1], the FPT algorithm for vertex cover cannot be extended already to the parameter distance to linear forest.

▶ **Theorem 4.20** (⋆). *Self-Deleting s-t-path is* W[1]*-complete parameterized by the distance to linear forest, even if* $\mu \leq 1$.

Dense parameters and μ combined

In the remainder of this section we focus on parameters whose bounded values together with the existence of a long path imply dense structure of the graph in some sense. As a warmup example, recall that Self-Deleting s-t-path is NP-hard on cliques (Corollary 3.9). Observe that there is always an f-conforming path on at most $\mu + 2$ vertices when the underlying graph is a clique, if there is any f-conforming path at all. This is because the vertex s cannot delete more than μ edges, so if the path is longer, then there is a shortcut that we can take. By plugging this upper bound into Theorem 4.17, we obtain the following:

▶ **Observation 4.21.** *Self-Deleting s-t-path is solvable in* $2^{O(\mu \log \mu)}(n+m)\log n$ *time on cliques.*

We build upon this idea. In general, vertices of a k-vertex path can delete at most $k\mu$ edges. On the other hand, if we consider only shortest (f-conforming) paths, any shortcut edge on the vertices of the path must be deleted as otherwise the path can be shortened, contradicting that it is the shortest path. We thus obtain the following:

▶ **Observation 4.22.** *Let* $(G = (V, E), f)$ *be a self-deleting graph and let* $P = (v_1, v_2, \ldots, v_k)$ *be a shortest* f*-conforming* v_1*-*v_k *path in* (G, f)*. Let* $G_P = G[V(P)]$ *be the subgraph of* G *induced by the vertices of* P*. Then* $|E(G_P)| \leq k\mu + k - 1 \leq k(\mu + 1)$.

We now utilize the results of Dvořák et al. [12]. They provide a lower bound on the number of edges in the input graph, given that it contains long (Hamiltonian) path. Recall that graphs containing a Hamiltonian path are also called *traceable*. By combining these bounds together with Observation 4.22, we obtain an upper bound on the length of a shortest f-conforming s-t path in the underlying graph in terms of μ and some structural parameter α, see Table 1 for an overview. Recall that $\widetilde{O}$ supresses polylogarithmic factors.

▶ **Lemma 4.23.** *Let* α *be a graph parameter monotone under taking induced subgraphs. Suppose there are global constants* C, D *such that any traceable graph* G *with* $n \geq C \cdot \alpha(G)$ *vertices contains at least* $D\frac{n^2}{\alpha(G)}$ *edges. Then Self-Deleting s-t-path can be solved in* $2^{\widetilde{O}(\alpha\mu)}(n+m)\log n$ *time. In particular it is* FPT *parameterized by* α *and* μ *combined.*

Proof. Let (G, f, s, t) be the input instance of Self-Deleting s-t-path and let P be a shortest f-conforming s-t path in G. Let $k = |V(P)|$ and let $G_P = G[V(P)]$ be the graph induced by vertices of P. By assumption on α, either $k < C \cdot \alpha(G_P) \leq C \cdot \alpha(G)$ (because α is monotone), or $|E(G_P)| \geq D\frac{k^2}{\alpha(G_P)}$. By Observation 4.22 we also have $|E(G_P)| \leq k(\mu+1)$. By combining these two bounds we obtain the bound $k \leq \frac{1}{D}\alpha(G_P)(\mu+1) \leq \frac{1}{D}\alpha(G)(\mu+1)$. By plugging $k = \max\{C \cdot \alpha(G), \frac{1}{D}\alpha(G) \cdot (\mu+1)\}$ into Theorem 4.17 we obtain the promised algorithm with running time $2^{O(\alpha\mu\log(\alpha\mu))}(n+m)\log n$. ◀

Table 1 Overview of the framework for parameterization by structural parameters α and μ combined. The lower bound column is the asymptotic lower bound on the number of edges for traceable graph with given parameter bounded by α proven by Dvořák et al. [12]. The third column indicates what is the implied upper bound on the length of any shortest f-conforming path in such graphs. The fourth column is the final running time of the algorithm using the better from Theorems 4.16 and 4.17. Note that α should be replaced by $\max\{\alpha, 1\}$. We write just α for readability purposes.

parameter α	lower bound	length of path	running time
distance to cograph	$n \log \frac{n}{\alpha}$	$\alpha 2^{O(\mu)}$	$2^{\alpha 2^{O(\mu)}}(n+m)\log n$
cluster vertex deletion number	$\frac{n^2}{\alpha}$	$O(\alpha\mu)$	$2^{\widetilde{O}(\alpha\mu)}(n+m)\log n$
neighborhood diversity	$\frac{n^2}{\alpha}$	$O(\alpha\mu)$	$2^{\widetilde{O}(\alpha\mu)}(n+m)\log n$
modular-width	$n \log_\alpha n$	$\alpha^{O(\mu)}$	$2^{\alpha^{O(\mu)}}(n+m)\log n$
maximum induced matching	$\frac{n^2}{\alpha}$	$O(\alpha\mu)$	$2^{\widetilde{O}(\alpha\mu)}(n+m)\log n$
shrub-depth	$n^{1+\frac{1}{2^\alpha-1}}$	$\mu^{2^{O(\alpha)}}$	$2^{\mu^{2^{O(\alpha)}}}(n+m)\log n$

We could prove an analogous version of Lemma 4.23 also with the functions $n \mapsto D \cdot n \log \frac{n}{\alpha}$, $n \mapsto D \cdot n \log_\alpha n$, or $n \mapsto D \cdot n^{1+\frac{1}{2^\alpha-1}}$ with different running time of the algorithm (see Table 1).

▶ **Theorem 4.24.** *Self-Deleting s-t-path is* FPT *w.r.t. α and μ combined where α is one of the following parameters: cluster vertex deletion number, neighborhood diversity, distance to cograph, modular-width, maximum induced matching, shrub-depth.*

Proof. We prove the theorem for cluster vertex deletion number, the rest is proved similarly by using suitable lower bound from Table 1. If $\mathrm{cvdn}(G) = 0$, the graph is a cluster and we can restrict ourselves to the clique where s and t lies and use Observation 4.21. Otherwise, if $\mathrm{cvdn}(G) \geq 1$, by the result of Dvořák et al. [12] any traceable graph G on $n \geq 4\,\mathrm{cvdn}(G)$ vertices contains at least $\frac{n^2}{16(\mathrm{cvdn}(G)+1)} \geq \frac{n^2}{32\,\mathrm{cvdn}(G)}$ edges. Invoke Lemma 4.23 for $D = \frac{1}{32}$ and $C = 4$ to obtain the desired algorithm. ◀

Domination Number. FPT algorithms for distance to cograph, modular-width, or maximum induced matching cannot be extended to an FPT algorithm for the diameter of the graph (and μ combined). We show that Self-Deleting s-t-path is para-NP-hard already for domination number and μ combined.

▶ **Theorem 4.25** ($\star$). *Self-Deleting s-t-path remains* NP*-hard even when the self-deleting graph (G, f) satisfies $\gamma(G) = \mu_f = 1$.*

5 Kernels

In this section, we show that while Self-Deleting s-t-path admits FPT algorithms for broad number of structural parameters with μ combined, it does not admit a polynomial kernel w.r.t. the vertex cover number and μ combined already in the class of 2-outerplanar graphs unless the polynomial hierarchy collapses. Moreover, there is no polynomial kernel in the class of cliques w.r.t. μ, but there is a linear Turing kernel w.r.t. μ in the class of cliques.

▶ **Theorem 5.1** (⋆). *Unless* NP $\subseteq$ coNP/$_{\text{poly}}$, *Self-Deleting s-t-path does not admit a polynomial kernel with respect to*

a) vcn *and* μ *combined, even on* 2*-outerplanar graphs;*

b) vi *even with* $\mu = 1$ *and on* 2*-outerplanar graphs;*

c) μ *on cliques.*

Proof Sketch for a). We provide an OR-cross-composition of 3Sat into Self-Deleting s-t-path parameterized by vcn and μ. The idea is to use Construction 3.1 for a formula with all $O(n^3)$ clauses with auxiliary *skip edges* that allow to pass a clause for free. Before s we prepend selector vertices for τ instances that will delete appropriate skip edges and thus modify the rest of the graph to look like the reduction for the given formula. Details can be found in the full version of this paper. ◀

▶ **Theorem 5.2** (⋆). *Self-Deleting s-t-path admits*

a) *an* $O(\text{fen})$ *kernel;*

b) *an* $O(\text{vcn})$ *kernel on outerplanar graphs;*

c) *a Turing kernel with* $O(\mu)$ *vertices on cliques.*

▶ **Corollary 5.3** (⋆). *Self-Deleting s-t-path can be solved in* $2^{O(\mu)}n^2$ *time if the underlying graph is a clique and* $2^{o(\mu)}\operatorname{poly}(n)$*-time algorithm on cliques violates ETH.*

6 Conclusion and open problems

We initiated a systematic study of complexity of finding a simple path in a self-deleting graph, which we call Self-Deleting s-t-path. While the problem is hard on very restricted graph classes, we were able to design FPT algorithm(s) parameterized by the solution size and structure of the deletion function. This further allowed us to design FPT algorithms for various structural parameters combined with the structure of the deletion function.

Our derandomization of the color-coding algorithm yields running time of $2^{O(k\log(k\mu))}\operatorname{poly}(n)$ (Theorem 4.17) or $2^{O(k\mu)}\operatorname{poly}(n)$ (Theorem 4.16). We were unable to derandomize it in a way to match the randomized running time of $2^{O(k\log\mu)}\operatorname{poly}(n)$ from Theorem 4.15. We conjecture that with a suitable pseudorandom object, there is a way to derandomize the algorithm into a deterministic $2^{O(k\log\mu)}\operatorname{poly}(n)$ time.

Our framework for FPT algorithms parameterized by k and μ together with lower bounds on the number of edges in traceable graphs with dense structure does not give optimal running times under ETH. For example, already for cliques, the framework gives running time $2^{O(\mu\log\mu)}(n+m)$ (Observation 4.21) which is not optimal (Corollary 5.3). Assuming ETH, we cannot obtain an algorithm for Shortest Self-Deleting s-t-path with running time $2^{o(k)\log\mu}\operatorname{poly}(n)$ (see Remark 3.5). Similarly, an algorithm with running time $2^{k\cdot o(\log\mu)}\operatorname{poly}(n)$ would imply that Shortest Self-Deleting s-t-path parameterized by k is in FPT (this is due to [4, Lemma 1]), which would then imply that FPT = W[1]. Note that Shortest Self-Deleting s-t-path becomes FPT w.r.t. k if $\mu \in O(\log n)$ by plugging into the algorithm from Theorem 4.17 and using the fact that $(\log n)^{g(k)}$ is fpt-time.

Can the algorithms for structural parameters and μ be improved to match the above ETH lower bounds? For example, is it possible to solve Self-Deleting s-t-path in deterministic $2^{O(\text{vcn}\log\mu)}\operatorname{poly}(n)$ time (note that we can achieve such a running time by a randomized algorithm)?

References

1 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the Association for Computing Machinery*, 42(4):844–856, July 1995. doi:10.1145/210332.210337.

2 J. Añez, T. De La Barra, and B. Pérez. Dual graph representation of transport networks. *Transportation Research Part B: Methodological*, 30(3):209–216, 1996. doi:10.1016/0191-2615(95)00024-0.

3 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernel bounds for path and cycle problems. *Theoretical Computer Science*, 511:117–136, 2013. doi:10.1016/J.TCS.2012.09.006.

4 Liming Cai and David W. Juedes. Subexponential parameterized algorithms collapse the W-hierarchy. In Fernando Orejas, Paul G. Spirakis, and Jan van Leeuwen, editors, *Automata, Languages and Programming, 28th International Colloquium, ICALP 2001, Crete, Greece, July 8-12, 2001, Proceedings*, volume 2076 of *Lecture Notes in Computer Science*, pages 273–284. Springer, 2001. doi:10.1007/3-540-48224-5_23.

5 Sarah Carmesin, David Woller, David Parker, Miroslav Kulich, and Masoumeh Mansouri. The Hamiltonian cycle and travelling salesperson problems with traversal-dependent edge deletion. *Journal of Computer Science*, 74:102156, 2023. doi:10.1016/J.JOCS.2023.102156.

6 Raffaele Cerulli, Francesca Guerriero, Edoardo Scalzo, and Carmine Sorgente. Shortest paths with exclusive-disjunction arc pairs conflicts. *Computers & Operations Research*, 152:106158, 2023. doi:10.1016/J.COR.2023.106158.

7 Jianer Chen, Xiuzhen Huang, Iyad A. Kanj, and Ge Xia. Linear FPT reductions and computational lower bounds. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 212–221. ACM, 2004. doi:10.1145/1007352.1007391.

8 Jianer Chen, Xiuzhen Huang, Iyad A. Kanj, and Ge Xia. Strong computational lower bounds via parameterized complexity. *Journal of Computer and System Sciences*, 72(8):1346–1367, 2006. doi:10.1016/J.JCSS.2006.04.007.

9 Ting Chen, Ming-Yang Kao, Matthew Tepel, John Rush, and George M. Church. A dynamic programming approach to de novo peptide sequencing via tandem mass spectrometry. *Journal of Computational Biology*, 8(3):325–337, 2001. doi:10.1089/10665270152530872.

10 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

11 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

12 Michal Dvořák, Dušan Knop, Michal Opler, Jan Pokorný, Ondřej Suchý, and Krisztina Szilágyi. Density of traceable graphs, 2025. arXiv:2506.22269.

13 Harold N. Gabow, Shachindra N. Maheswari, and Leon J. Osterweil. On two problems in the generation of program test paths. *IEEE Transactions on Software Engineering*, 2(3):227–231, 1976. doi:10.1109/TSE.1976.233819.

14 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.

15 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001. doi:10.1006/JCSS.2001.1774.

16 Ronald F. Kirby and Renfrey B. Potts. The minimum route problem for networks with turn penalties and prohibitions. *Transportation Research*, 3(3):397–408, 1969. doi:10.1016/S0041-1647(69)80022-5.

17 Petr Kolman and Ondřej Pangrác. On the complexity of paths avoiding forbidden pairs. *Discrete Applied Mathematics*, 157(13):2871–2876, 2009. doi:10.1016/J.DAM.2009.03.018.

18 Jakub Kováč. Complexity of the path avoiding forbidden pairs problem revisited. *Discrete Applied Mathematics*, 161(10):1506–1512, 2013. doi:10.1016/j.dam.2012.12.022.

19 K. Krause, R. Smith, and M. Goodwin. Optimal software test planning through automated network analysis. In *Proceedings of the IEEE Symposium on Computer Software Reliability*, pages 18–22, 1973.

20 Stefan Szeider. Finding paths in graphs avoiding forbidden transitions. *Discrete Applied Mathematics*, 126(2):261–273, 2003. doi:10.1016/S0166-218X(02)00251-2.

21 Piotr Wojciechowski, K. Subramani, and Alvaro Velasquez. Reachability in choice networks. *Discrete Optimization*, 48:100761, 2023. doi:10.1016/j.disopt.2023.100761.

22 Piotr Wojciechowski, M. Williamson, and K. Subramani. On the analysis of optimization problems in arc-dependent networks. *Discrete Optimization*, 45:100729, 2022. doi:10.1016/j.disopt.2022.100729.

23 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. doi:10.1109/TKDE.2016.2594065.

24 Hananya Yinnone. On paths avoiding forbidden pairs of vertices in a graph. *Discrete Applied Mathematics*, 74(1):85–92, 1997. doi:10.1016/S0166-218X(96)00017-0.

25 Athanasios K. Ziliaskopoulos and Hani S. Mahmassani. A note on least time path computation considering delays and prohibitions for intersection movements. *Transportation Research Part B: Methodological*, 30(5):359–367, 1996. doi:10.1016/0191-2615(96)00001-X.

Space-Efficient Depth-First Search via Augmented Succinct Graph Encodings

Michael Elberfeld
THM, University of Applied Sciences Mittelhessen, Gießen, Germany

Frank Kammer
THM, University of Applied Sciences Mittelhessen, Gießen, Germany

Johannes Meintrup
THM, University of Applied Sciences Mittelhessen, Gießen, Germany

Abstract

We call a graph G separable if a balanced separator can be computed for G of size $O(n^\epsilon)$ with $\epsilon < 1$. Many real-world graphs are separable such as graphs of bounded genus, graphs of constant treewidth, and graphs excluding a fixed minor. In particular, the well-known planar graphs are separable. We present a succinct encoding of separable graphs G such that, after the encoding is computed, any number of depth-first searches (DFS) can be performed from any given start vertex, each in $o(n)$ time and $o(n)$ bits in the word RAM model. After the execution of a DFS, the succinct encoding of G is augmented such that the DFS tree is encoded inside the encoding while maintaining succinctness. Afterward, the encoding provides common DFS-related queries in constant time. These queries include queries such as lowest-common ancestor of two given vertices in the DFS tree or queries that output the lowpoint of a given vertex in the DFS tree. Furthermore, for planar graphs, we show that the succinct encoding can be computed in $O(n)$ bits and expected linear time, and a compact variant can be constructed in $O(n)$ time and bits. For other separable graph classes $\mathcal{G}$ the runtime and space usage depends on the specific algorithms used to find balanced separators in graphs of $\mathcal{G}$.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis; Theory of computation → Graph algorithms analysis

Keywords and phrases Depth-First Search, Succinct, Space Efficient, Separable Graphs, Planar Graphs, Table Lookup, r-Division

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.29

Related Version *Extended Version*: `https://arxiv.org/abs/2504.19547`

1 Introduction

Depth-first search (DFS) in graphs forms the backbone of algorithms for a number of applications like finding vertex or edge cuts. A depth-first search implicitly computes a tree (or forest), over the vertices of the graph that is called a *DFS tree* (or forest), which has crucial structural properties that are commonly used for applications. DFS that runs in linear time in the number of vertices and edges utilizes two folklore data structures: a stack to keep track of the current paths into the graph and an adjacency list to efficiently iterate over the neighbors of a given vertex. Standard implementations of this approach use $\Theta(n \log n)$ bits of space for the stack and $O(\log n)$ bits for each vertex identifier, where n refers to the number of vertices in a given graph. Lowering the space requirements for depth-first search to $O(n)$ bits while still maintaining a (nearly) linear runtime was the aim of a series of works during the last years: one of the first algorithms is due to Asano et al. [3] who reduced the space to $O(n)$ bits, but increased the runtime to $O(m \log n)$, where m refers to the number of edges in a given graph. After several improvements [4, 7, 9] to this result, Hagerup [13, Theorem 5.4] presented the current state-of-the-art algorithm using $O(n)$ bits and $O(n + m \log^* n)$ time.

© Michael Elberfeld, Frank Kammer, and Johannes Meintrup;
licensed under Creative Commons License CC-BY 4.0

In many circumstances, we do not need to be able to handle input graphs of any kind. Instead, the input graphs have a special structure that we can utilize to devise algorithms that are more efficient than the ones handling the general case. Typical examples are planar graphs, which can be drawn in the plane without edge crossings, or generalizations of it like graphs of bounded genus or graphs excluding a fixed minor. What these graphs have in common is that they are sparse, meaning $m = O(n)$. As there exists a DFS that uses $O(n + m)$ time and bits on general graphs, as stated by Hagerup [13, Theorem 4.1] and Banerjee et al. [4, Lemma 2], we can execute a DFS with $O(n)$ time and bits on sparse graphs. If we shift our attention to $o(n)$-bit DFS algorithms, we can observe some strong indications from complexity that no sublinear space polynomial time algorithm can exist for a special variant of DFS, called *lexicographical DFS* [3, 9]. A concrete sublinear space algorithm for this problem restricted to planar graphs has an unwieldy massive polynomial runtime [18], due to the reliance on the logspace reachability result of Reingold [27].

As mentioned above, computing a DFS is often just the beginning or part of an algorithm solving more involved graph problems like computing biconnected and strongly-connected components. Typically, these applications of DFS store certain meta information that is computed while traversing the graph. Examples are *pre-order numbering* that numbers the vertices in the order of their exploration [28], and *lowpoints* that help to identify vertex and edge cuts [17]. If we want to extend existing approaches for computing DFS that are efficient in both space and time to this more general setting, we can not easily store the meta information of a vertex using any standard encoding. Again, this would use $\Theta(\log n)$ bits for each vertex and, hence, creates a total memory footprint of $\Theta(n \log n)$ bits. While there exists space-efficient techniques for, e.g., computing pre-order numbers or lowpoints [7, 13, 20], they require significantly more resources than the space-efficient DFS variants, e.g., the algorithm Chakraborty et al. [7] for computing lowpoints uses $\Theta(n)$ bits and $\Theta(m \log n \log \log n)$ time.

In this paper we present both (1) a data structure that is able to keep meta information we compute for the vertices of a graph space-efficiently and (2) an approach for computing DFS traversals space-efficiently as well as extensions to some common applications. Our results are applicable to classes of separable graphs which have balanced separator size $O(n^\epsilon)$ for some $\epsilon < 1$. This covers, in particular, planar graphs, graphs of bounded genus and graphs excluding a fixed minor.

Data Structure: Nested Divisions and Augmentations. An encoding of an element from a universe U is called *succinct* (*compact*) if it uses $Z + o(Z)$ ($Z + O(Z)$) bits where $Z = \log_2 |U|$. In this paper we present a succinct encoding for separable graphs (precisely defined in Section 2.1), such as planar graphs, that provides the following functionalities. After the encoding is initialized, a DFS can be executed (from any given start vertex) in $o(n)$ time and $o(n)$ bits and afterward information of the executed DFS can be queried in constant time. If needed, a new DFS can be computed at any given time. Such information includes the pre-order numbering and lowpoints of a vertex, and lowest-common ancestor of two vertices. Furthermore, for planar graphs, we show that the succinct encoding can be computed in expected linear time with $O(n)$ bits used during the construction step, and a compact variant can be constructed in $O(n)$ time and bits.

The data structure we present, called *succinct nested division*, extends a succinct encoding of separable graphs by Blelloch and Farzan [6]. The encoding of Blelloch and Farzan is built on dividing the input graph into smaller "mini pieces". Mini pieces are, in turn, divided into even smaller "micro pieces" that are small enough such that relevant structural information about them can be pre-computed and kept in a lookup table. The key property of this

approach is that there are only few "boundary vertices" in each piece that are contained in multiple pieces. This suggests algorithms that, like the data structure itself, switch between these three levels. In fact, different algorithmic approaches are needed for different levels to maintain our target space and time bounds.

We augment the encoding of Blelloch and Farzan with additional data structures and show that this can be used to design more complex queries than the standard graph access queries provided by them. Intuitively, the boundary vertices act like relays since any interaction between two non-boundary vertices in different pieces must necessarily pass through them. Consequently, long-range dependencies (e.g., long paths), are efficiently mediated by the sparse set of boundary vertices. Hence, pieces can often be considered in isolation with few interactions between them, and all interactions between vertices of different pieces are communicated via boundary vertices. To capture the idea we intuitively call a property π *strongly local* if, for all non-boundary vertices u, $\pi(u)$ can be evaluated only with information stored with the boundary vertices together with a small amount of information that is directly encoded in P, typically for us there exists a boundary vertex v of P such that $|\pi(u) - \pi(v)| = O(|P|)$.

For circumventing the trivial lower bound of $\Omega(n \log n)$ on the number of bits required for encoding members of a family of labeled graphs with n vertices, we work with so-called unlabeled graphs. This means that we are able to choose our own vertex labels for the encoding so that we can construct a succinct encoding for separable graphs that uses $\Theta(n)$ bits [6]. Such a setting is very common, e.g., used by Blelloch and Farzan's original encoding [6] as well as other works [1, 2, 8, 11, 19, 22, 25].

Algorithms: Depths-First Search and Applications. Our succinct encoding for separable graphs is constructed such that a DFS can be performed directly on the encoding from any given starting vertex. Afterward, various queries regarding the DFS traversal are available. In particular, we directly provide the necessary queries that, e.g., Hopcroft and Tarjan's biconnected-component algorithm [17], or Schmidt's algorithm for chain decompositions [28] require. We effectively provide an interface that allows us to implement typical standard algorithms without the need to design specialized techniques. The following theorem summarizes our main results. Note that the runtime and bits are sublinear once the encoding is computed. The operations presented in it are only a set of examples of queries that have the previously outlined intuition of strongly local, chosen such that the previously mentioned standard algorithms can be executed directly. Without loss of generality, we assume that all graphs we work with are connected. If a given graph is not connected, one can apply all our results to each connected component separately.

▶ **Theorem 1.** *Let G be a separable graph. There exists a succinct encoding $\mathcal{D}$ of G that provides neighborhood iteration, adjacency and degree queries in constant time per element output. Additionally, it provides the operation* $\mathrm{dfs}(u)$ *that executes a DFS in $O(n/\operatorname{poly}(\log\log n)) = o(n)$ time and $o(n)$ bits from a given start vertex u such that at all times the encoding remains succinct. After the execution of a DFS the following queries are available in constant time for the last computed DFS tree.*

- $\mathrm{children}(u)$ *iteration over all children of u.*
- $\mathrm{parent}(u)$*: output the parent of u.*
- $\mathrm{pre}(u)/\mathrm{post}(u)$*: return the pre-/post-order number of vertex u.*
- $\mathrm{num\text{-}descendants}(u)$*: return the number of descendants of u.*
- $\mathrm{depth}(u)$*: return the depth of u.*
- $\mathrm{lowpoint}(u)$*: return the lowpoint of u.*
- $\mathrm{lca}(u, v)$*: return the lowest-common ancestor of vertices u and v.*

Replacing succinct with compact in Theorem 1, and additionally restricting G to be a planar graph, the encoding can be constructed with $O(n)$ bits and in $O(n)$ time. The change from succinct to compact is due to our wish to avoid the use of a certain succinct dictionary data structure of Raman et al. [26] relying on hashing functions that can only be computed in expected linear time. When we replace the dictionary with a simpler variant of Baumann and Hagerup [5], we obtain a compact encoding that can be computed much more easily. When we are fine with expected linear time, we can use the dictionary of Raman et al., and thus can construct the succinct encoding in $O(n)$ bits and expected $O(n)$ time. Subsection 3.3 presents the details regarding the construction steps. For other graph classes the construction time and bits depend on the runtime of the respective separator algorithms used as subroutines to divide the input graph.

▶ **Corollary 2.** *Let G be a planar graph. There exists a compact (succinct) encoding of G that can be constructed in $O(n)$ time (expected time) and $O(n)$ bits that provides the same functionalities as the encoding of Theorem 1.*

Related Work on Graph Divisions. Recursive divisions have been used as the basis for algorithms and data structures for decades, commonly based on the so-called r-division for planar graphs, defined precisely in the next section. These divisions build on a recursive application of the linear-time separator algorithm of Lipton and Tarjan [24], and the improvement of Goodrich [12] who showed that the entire recursively application can be executed in linear time – a standard approach would use $O(n \log n)$ time. Recursive divisions have been successfully applied to algorithmic problems such as maximum flow and minimum cut and all-pairs shortest path [10]. Applications in the field of data structures include decremental data structures for connectivity [15, 16] where decremental refers to modifications of a graph that remove vertices or edges. None of the mentioned results have a focus on space-efficiency, and internally they use other techniques compared to our approach.

Structure of the Paper. Section 2 details our first contribution, the nested-division data structure and augmented variants of it. Subsequently, in Section 3, we present the sublinear-space and time execution of DFS and related applications, executed directly on our data structure.

2 Nested Divisions

The present section contains an overview of what we view as a nested division, beginning with the graph-theoretic properties they provide us with (Subsection 2.1), and continuing how Blelloch and Farzan used nested divisions as a data structure to encode separable graphs (Subsection 2.2). Finally, we extend the section with our ideas of abstract augmentations in nested divisions we provide to allow complex queries (Subsection 2.3). We use the notation $[i]$ for any natural number i to refer to the set $\{1, \ldots, i\}$.

2.1 Graph-Theoretic Properties of Nested Divisions

We describe a slightly generalized variant of the well-known concept of r-divisions sketched in the introduction. We start with a small set of definitions. A *balanced separator* of a graph $G = (V, E)$ is a set of vertices $S \subset V$ such that each connected component of $G[V \setminus S]$ contains at most a constant fraction of the vertices. The exact constant does not matter to us. We call a class of graphs $\mathcal{G}$ that is closed under taking vertex induced subgraphs

$O(n^\epsilon)$-*separable* if there exists a constant ϵ with $0 < \epsilon < 1$, such that each $G = (V, E) \in \mathcal{G}$ has a balanced separator $S \subset V$ of size $O(n^\epsilon)$. We simply say that a graph or class of graphs is *separable* if the size of the balanced separator is of no particular concern to us.

▶ **Definition 3** (Relaxed Division). *Let G be a graph belonging to an $O(n^\epsilon)$-separable class of graphs for some $0 < \epsilon < 1$. An* (α, r)-relaxed division *is a decomposition of G into a collection $\mathcal{P}$ of subgraphs, called* pieces*, satisfying:*

1. *Each piece is a subgraph with at most r vertices such that there are $\Theta(n/r)$ pieces in total.*
2. *Every edge is assigned to exactly one piece (containing both endpoints of the edge).*
3. *Each piece has $O(\alpha r^\epsilon)$* boundary vertices*, which are vertices shared between different pieces.*

We call α the relaxation *and r the* piece size*. Edges between two boundary vertices are called* boundary edges*, edges between two non-boundary vertices* non-boundary edges *and all other edges* transitional edges*.*

See Figure 1 for a sketch of such a division. The reason we introduce the relaxation α, is due to our desire to achieve space-efficiency when algorithmically constructing a division. There exists an algorithm that computes a $(\log n, r)$-relaxed division via recursive separator searches that uses only $O(n)$ bits and, for planar graphs, $O(n)$ time [21]. If the construction step of our encoding must not be space-efficient, one can of course use a standard non-space efficient algorithm [12, 23], and therefore have a relaxation factor of $\alpha = 1$; for our application a factor of $\alpha = \log n$ allows for a more (space-)efficient computation of our final encoding, buy when using the encoding makes no difference. Common applications of divisions use them in a recursive fashion, and so do we. In particular, we construct for each piece of some (α, r)-relaxed division a $(1, \tilde{r})$-relaxed division where $\tilde{r} \ll r$. For this, we define a *nested division* for a separable graph G as follows.

▶ **Definition 4** (Nested Division). *Let G be a graph belonging to an $O(n^\epsilon)$-separable class of graphs for some $0 < \epsilon < 1$. An* $(\alpha, r, \tilde{r})$-nested division *of G is an (α, r)-relaxed division of G into pieces $\mathcal{P} = \{P_1, P_2, \ldots\}$ such that for each piece $P_i \in \mathcal{P}$ we have a $(1, \tilde{r})$-relaxed division $\mathcal{P}_i$. We call pieces of $\mathcal{P}$* mini pieces *and pieces of some $\mathcal{P}_i$* micro pieces*.*

If we have an $(\alpha, r, \tilde{r})$-nested division for a graph G each vertex can be viewed as being part of three *levels*:

- *graph*: Each vertex is part of V.
- *mini*: Each vertex of V is assigned to one or more mini pieces.
- *micro*: Each vertex of V is assigned to one or more micro pieces.

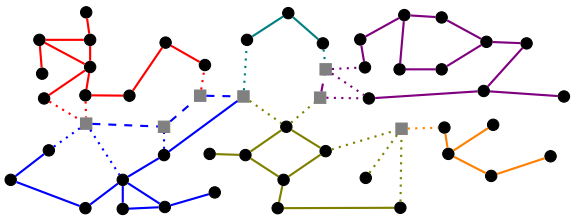


Figure 1 A sketch of a division of graph into 6 pieces of at most 13 vertices. The edges of each piece are colored with a distinct color. Boundary vertices are gray squares, and non-boundary vertices black circles. Non-boundary edges are solid, transitional edges dotted, and boundary edges dashed.

Given an $(\alpha, r, \tilde{r})$-nested division for a graph G we can categorize all vertices as follows. We call a vertex *mini (micro) boundary* if it is a boundary vertex in a mini (micro) piece, and *mini (micro) non-boundary* if it is not a boundary vertex in a mini (micro) piece. Edges are categorized similarly: an edge is called *mini (micro) boundary edge* if both endpoints are mini (micro) boundary vertices, *mini* (micro) non-boundary edge if both endpoints are mini (micro) non-boundary vertices, and *mini (micro) transitional edge* otherwise. We routinely drop the specifier mini or micro if we make statements that apply to both mini and micro levels. We are interested in bounding the number of total occurrences of mini (micro) boundary vertices among all mini (micro) pieces, referred to as *mini (micro) duplicates*. Let $\mathcal{P}$ be an (α, r)-relaxed division constructed for an $O(n^\epsilon)$-separable graph G. As each piece has $O(r^\epsilon)$ boundary vertices, the total number of duplicates is $O((n/r)(\alpha r^\epsilon))$. Thus, for a nested division we have $O(\alpha n/r^{1-\epsilon})$ (duplicates of) mini boundary vertices, and $O((n/r)(r/\tilde{r})(\tilde{r}^\epsilon)) = O(n/\tilde{r}^{1-\epsilon})$ (duplicates of) micro boundary vertices. We refer to a more detailed description of the reasoning behind these bounds to Blelloch and Farzan [6]. We summarize it in the following lemma.

▶ **Lemma 5** ([6]). *Let $\mathcal{D}$ be a $(\alpha, r, \tilde{r})$-nested division constructed for an $O(n^\epsilon)$-separable graph G with relaxation parameters α, and piece sizes r and $\tilde{r}$. Then there are $O(\alpha n/r^{1-\epsilon})$ (duplicates of) mini boundary vertices and $O(n/\tilde{r}^{1-\epsilon})$ (duplicates of) micro boundary vertices.*

For all of our use cases of $(\alpha, r, \tilde{r})$-nested divisions we have $\alpha = O(\log n)$, $r = \text{poly}(\log n)$ and $\tilde{r} = \text{poly}(\log \log n)$, with the exact polynomial chosen such that we have $O(n/\text{poly}(\log n))$ (duplicates of) mini boundary vertices, and $O(n/\text{poly}(\log \log n))$ (duplicates of) micro boundary vertices. For the remainder of our paper, when we refer to a *nested division* we refer to a $(\log n, \text{poly}(\log n), \text{poly}(\log \log n))$-nested division.

2.2 Nested Divisions as Data Structures

As we now describe concrete data structures, we fix our model of computation to the word RAM model with word size $\Omega(\log n)$. For the rest of the section, let G be a separable graph and assume that a nested division is given for G. We begin by outlining the most basic functionality we require for a nested division when used as a data structure to encode a graph. Blelloch and Farzan presented a succinct encoding for unlabeled separable graphs that effectively uses a nested division at its core, and the framework we present in this subsection is based on their work [6], defined slightly more general to allow for our later augmentations. While they do not use an α-relaxation factor for the initial (α, r)-relaxed division of the nested division (they use their own variation of a standard recursively constructed r-division), their framework works with $\alpha = \log n$ without any modifications. Their general idea is to succinctly encode the nested division, and encode each piece at the micro level as an index into a lookup table. Boundary vertices are encoded via additional data structures that use standard non-space efficient techniques, e.g., using arrays and lists. Combined with a succinct bidirectional mapping that maps a vertex to its different occurrences in each level (graph, mini and micro) they realize neighborhood, adjacency and degree queries.

To describe the functionality of these bidirectional mappings, we introduce a labeling of pieces and vertices. Each mini piece $P_i \in \mathcal{P}$ is uniquely identified by an id i. Analogously, the relaxed division constructed for each P_i is identified as $\mathcal{P}_i$. Each micro piece $P_{i,j}$ is identified by a tuple (i, j) with j indicating it is the piece with id j, i.e., piece $P_{i,j} \in \mathcal{P}_i$. Each vertex $u \in V$ has a *graph label* assigned from $[n]$, a *mini label* u_i assigned from $[r]$ in each mini piece P_i it is contained, and a *micro label* assigned from $[\tilde{r}]$ in each micro piece $P_{i,j}$ it is contained. When we talk about a vertex $u \in V$ we always assume that u is identified by its graph label.

The mappings that are provided are as follows: given a vertex $u \in V$ output the tuples (i, u_i) such u_i is the mini label of u in the piece P_i. Analogously for a given tuple (i, u_i) where i refers to a mini piece P_i and u_i is the label of some vertex of P_i, output the tuple $(j, u_{i,j})$ where $u_{i,j}$ is the micro label of vertex u_i in a micro piece $P_{i,j}$. Note that each of these queries can output more than one element if the vertex is a boundary vertex at the respective level. For less verbose writing we implicitly assume that we always have access to all mappings outlined in this paragraph and only make distinctions between the different types of labeling if necessary. We refer to the set of all outlined mappings as *translation mappings*.

▶ **Lemma 6** (Succinct Nested Division [6]). *Let $G \in \mathcal{G}$ be a graph and $\mathcal{G}$ a separable class of graphs. There exists a succinct data structure $\mathcal{D}$ of G that represents a nested division of G such that level mapping queries and level graph queries are provided in constant time.*

2.3 Augmenting Nested Divisions

In the following we want to augment the nested division of Lemma 6 with various capabilities. For this we analyze the runtime and memory budget we have when our goal is to run some algorithms in $o(n)$ time and with $o(n)$ bits, followed by outlining necessary techniques. As stated, we construct our nested divisions with $r = \text{poly}(\log n)$, $\tilde{r} = \text{poly}(\log \log n)$ and $\alpha = \log n$. This means that for each of the $O(n/\text{poly}(\log n))$ mini boundary vertices we have a memory budget of $\text{poly}(\log n)$ bits. Analogously, we have a budget of $\text{poly}(\log \log n)$ bits for each of the $O(n/\text{poly}(\log \log n))$ micro-boundary vertices. This means, we have a budget of $\text{poly}(\log n)$ bits for each mini piece and $\text{poly}(\log \log n)$ for each micro piece. These bounds allow us to use more space than what is needed for our applications, as we only require $O(\log n)$ bits per mini boundary vertex (and mini piece), and $O(\log \log n)$ bits per micro boundary vertex (and micro piece) for our applications. We have the same budget for the runtime as we do for memory, but for our application only require (amortized) constant time per boundary vertex.

For micro pieces we use a table lookup technique for constant time computations to subproblems. Micro pieces are so tiny, that any query we require can be described by a constant number of words in the word RAM model. We assume that we have a lookup table available that lists all graphs with at most $\tilde{r}$ vertices of the separable graph class $\mathcal{G}$ we work with. The table lists all graphs modulo isomorphism, i.e., no two entries are isomorphic. This is not enough for us, as we require additional data for the boundary vertices. Instead, we require the graphs of the lookup table to be *partially augmented*, meaning for at most $b = O(\tilde{r}^\epsilon)$ vertices of each graph G' of the lookup table we store a bit string (called a *partial vertex augmentation*) of some fixed length $\ell = O(\log \tilde{r})$, and an additional bitstring of length $O(b\ell)$ (called a *partial graph augmentation*).

Using a lookup table that lists all partially augmented graphs with at most $\tilde{r}$ vertices, we can realize what we refer to as *swapping indices*. Each micro piece is encoded as an index into the lookup table referencing some graph G' together with its partial augmentations. When we want to update values stored for (the micro boundary vertices of) the micro piece we concretely do this by changing the index we store. Let $P_{i,j}$ be some micro piece encoded as an index x into the lookup table. To update an augmented value in the micro piece, e.g., color a boundary vertex, we can replace the index x with a different index y. See Figure 2 for an example. We have to take care while constructing the table that any changes of the partial augmentation (i.e., the values stored for boundary vertices) does not change the internal labels of the graph, i.e., the names of the vertices stay the same independent of the changes we make to the data stored for boundary vertices. For the rest of our paper we always assume that we have access to such a lookup table.

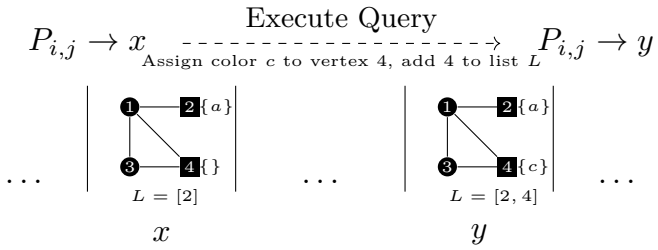


Figure 2 Initially the micro piece $P_{i,j}$ is stored as an index x of the table lookup, sketched below with two entries explicitly shown. The graph shown in the table has regular vertices shown as circles, and the vertices for which we can store binary strings shown as rectangles. We have a list L for each graph of the table that has enough capacity to store some values. We execute a query that takes as input the index x, the vertex label 4 and color c to assign to vertex 4. It returns the index y of the lookup table such that y stores the state that represents the execution of this query when applied to the graph stored at index x. We then swap the index x stored for $P_{i,j}$ with the index y in our succinct encoding.

3 Depths-First Search in Sublinear Time and Space

We present our techniques for executing a DFS directly on the succinct encoding. First, we show how the DFS is computed such that a so-called palm tree is encoded inside the nested division (Subsection 3.1), followed by our description of how to provide queries for so-called *strongly local* values (Subsection 3.2). As examples for strongly local values we describe the technical realization of providing queries regarding the meta-information of a DFS, e.g., pre-order numbering, lowpoints, lowest-common ancestors, and more. We end the section with a description of how our main results can be realized.

For this section, let $G = (V, E)$ be a separable graph given as a succinct nested division $\mathcal{D}$ (Lemma 6). We begin with the description of how to execute a DFS to compute a *palm tree* T that is a directed version of G such that the edges E are directed according to the DFS traversal, and by their direction partitioned into two sets, tree edges and back edges. Effectively, a palm tree is a DFS tree with additional back edges. Each vertex $u \in V(G)$ is assigned a *pre-ordering number* $\mathrm{pre}(u)$ indicating at which point in time it was traversed for the first time. Analogously, a *post-ordering number* $\mathrm{post}(u)$ is indicating at which point in the time DFS backtracked past the vertex u (i.e., all children of u have been fully explored). All edges (u, v) of the palm tree are directed such that, for a tree edge $\{u, v\}$ of the DFS tree, it holds that $\mathrm{pre}(u) < \mathrm{pre}(v)$, and for all other edges $\{u, v\}$ (the back edges), it holds that $\mathrm{pre}(v) < \mathrm{pre}(u)$. In the next subsection we compute some palm tree T directly on $\mathcal{D}$ and afterward provide constant time queries regarding T such as iteration over all children of a vertex u . If we are only interested in the directed tree edges of T, we refer to it as the DFS tree.

3.1 Iterator based DFS to compute a Palm Tree

In addition to the standard stack-based approach for implementing a DFS, there is also an iterator approach that stores for each vertex an iterator of the adjacency list. We implement the second variant. Let T be the palm tree constructed by some DFS traversal. See Figure 3 for an example of the following description. We say T *enters* a piece P if the there exists tree edges (u, v) and (v, w) with only the edge $\{v, w\}$ part of the piece P, or if edge $\{v, w\}$ is part of the piece P, (v, w) is a tree edge, and v is the root of T. We then call v an *entry vertex of*

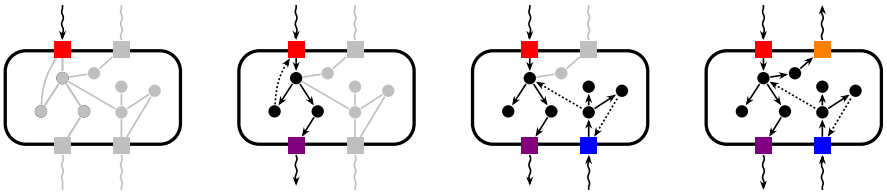


Figure 3 The figure shows four DFS states of some piece P. In the first state, no vertex inside P is visited yet, and we have just arrived at a boundary vertex of P (colored red). We then advance the DFS until the next boundary vertex is visited, colored violet. For micro pieces, we advance it via one query to the lookup table. We later arrive at the same piece again, this time via the blue boundary vertex. We enter the piece, and encounter the situation of the null-exit, thus we fully explore all vertices belonging to the subtree rooted at the blue vertex. Finally, we backtrack to the piece P via the violet vertex, and find a part of the piece not yet visited. We advance the state until the orange boundary vertex is visited and leave the piece again. The entry-exit pairs of the piece are (•, •), (•, `null`) and (•, •). Tree edges are shown as solid lines, and back edges as dotted lines.

P. Furthermore, we say T *exits* P if there exist tree edges (u, v) and (v, w) with only $\{u, v\}$ part of P and $\{v, w\}$ not part of P. We then call v an *exit vertex of* P. We say T *starts* at a piece P if the tree edge (u, v) exists with u the root of T, v is the first vertex visited after u and $\{u, v\}$ is part of P. Note that the entry of a piece is by definition a boundary vertex of some piece P, or the root of T. For us there is no difference between the two cases of entry vertex (i.e., boundary vertex or root). We refer to an exit vertex together with its matching entry vertex as an *entry-exit pair*. For easier description we refer to an entry vertex that has no matching exit vertex as part of an entry-exit pair where the entry is mapped to some special symbol, called the *null exit*. The following observation directly follows.

▶ **Observation 7.** *Let b be the number of (micro/mini) boundary vertices contained in a (micro/mini) piece P and T any palm tree. Then there are $O(b)$ entry-exit pairs associated with P.*

Let P be some micro piece stored as an index to the lookup table. We define a *partial DFS state of* P as a coloring of the vertices of P with the colors unvisited (white), currently visited (gray) and finished (black), and a constant number of (possibly empty) ordered sets of entry-exit pairs, ordered by, e.g., the pre-/and post-order numbering of their respective entry vertex, with ties being broken arbitrarily.

By the capabilities of the table lookup outlined in the previous section, an index of the lookup table is able to encode this information. The idea is that, when the DFS enters a micro piece P, categorized by some index x of the lookup table encoding P together with its partial DFS state, we obtain in $O(1)$ time an index y of the lookup table encoding P together with the next partial DFS state. We then swap out the index x with the index y. Note that this query depends on: the piece P, the coloring of the vertices of P, the ordered set of entry-exit pairs and the vertex we used to just enter the piece P (or the vertex we just backtracked to). This information is enough to obtain a new DFS state that correctly advances the DFS through P while considering the current partial DFS state. Since such an advancement results in the DFS traversing up to (or backtracking to) the next micro boundary vertex we obtain a new entry-exit pair in $O(1)$ time. There are multiple possible

viable partial DFS states that one could obtain by this operation, but we store one fixed state per entry of the lookup table, which is all that we require. See Figure 3 again for a visualization of how the DFS progresses through a piece. The change from one state of the figure to the next is done in $O(1)$ time for micro pieces.

A final note considers marking vertices as gray or black in micro pieces. When we visit a micro boundary vertex u, we must color u in all micro pieces $P_{i,j}$ that contain u (as micro label $u_{i,j}$). The translation mappings allow us to iterate over all micro pieces that contain $u_{i,j}$ in constant time per element output. For each such element output we must switch out the respective index of the micro pieces via the lookup table. This is done exactly twice (white to gray, and gray to black) per duplicate of a micro boundary vertices, and thus the time is linear in the number of total micro boundary vertices.

We now describe the data structures required for implementing the DFS. We begin with a description of how to implement an iterator over the neighborhood that can be paused and resumed. Note that while the nested division $\mathcal{D}$ provides neighborhood iteration (Lemma 6), it is not assumed that this iteration can be paused and continued at a later point in time without starting the process from the beginning. First, we describe this process for micro boundary vertices u_i in some piece P_i. For each such u_i store an array A containing the indices of each micro piece $P_{i,j}$ that contains u_i (as micro label $u_{i,j}$). An iterator for u_i now consists of an index j of A together with an index into the adjacency array of $u_{i,j}$ in $P_{i,j}$. Note that the array A contains many entries for each micro boundary vertex. By Lemma 5 this is asymptotically no problem, as the number of entries in all arrays A depends linearly on the number of micro boundary vertices. The arrays A can be built in sublinear time and space by iterating over all micro pieces, and inside each micro piece iterating over all micro-boundary vertices. We store the analogous data structure for all mini boundary vertices. We now must store the state of each iterator during the DFS. Construct an array storing for each boundary vertex the state of the iterator over u's neighborhood together with u's parent in the DFS tree and the color visited/unvisited. This uses $O(\log n)$ bits per mini boundary vertex. For each mini piece P_i we construct arrays storing the respective information of micro boundary vertices. As each vertex u (identified via mini label u_i) in P_i has a degree of $O(r)$ and must have its parent in P_i (identified by some mini label $v_i \in [r]$), this information can be stored with $O(\log r) = O(\log \log n)$ bits per micro boundary vertex. All other vertices are handled by the lookup table. All other data structures use $o(n)$ bits.

The DFS naturally computes the palm tree, but it remains to show how to store it such that afterward we can execute common queries such as iteration over all children (i.e., incident tree edges), or iteration over all back edges that start/end at a given vertex. We describe this for iteration over all children of a vertex, the other queries are realized in the exact same fashion. For micro non-boundary vertices these queries can directly be provided by the table lookup, or by recursive structure. We thus focus on the boundary vertices. For each mini boundary vertex u we maintain a list L containing the ids of all mini pieces P such that u has a child in P. Iteration over the children of u can then be expressed as an iteration over L, and for each $i \in L$ we output all children of u in P_i. Concretely this is done by outputting all children of u_i (the mini label of u in P_i) as their respective mini labels, and then translating them to their global labels. For micro boundary vertices, analogous structures are built. The space analysis is analogous to the space analysis of the previous paragraph, i.e., it uses $o(n)$ bits total. We summarize the results in the next lemma.

▶ **Lemma 8.** *Let G be a separable graph given as a nested division $\mathcal{D}$. In $O(n/\operatorname{poly}(\log \log n)) = o(n)$ time and $o(n)$ bits we can execute a DFS on $\mathcal{D}$ and store the resulting palm tree T such that the following queries are available for any vertex u, in constant time (per element output).*

- *Iteration over the children v of u in T, ordered by* $\mathrm{pre}(v)$.
- *Iteration over all back edges starting/ending at u in T.*
- *Output the parent u.*

3.2 Meta Information of a Depth-First Search

During the computation of a DFS it is common to store various values that relate to the traversal such as the pre-order $\mathrm{pre}(u)$ of a vertex u, or more complex values such as the *lowpoint* of a vertex u, defined as the lowest numbered (by pre-order numbering) vertex reachable via a path starting at u that consists of zero or more tree edges and at most one back edge. We can augment $\mathcal{D}$ with this information as follows. We start to describe the computation of the pre-ordering number $\mathrm{pre}(u)$. Post-order can be implemented in an analogous way. Assume that we have constructed the palm tree of Lemma 8.

Pre-order numbering. We begin with an observation regarding the pre-order numbering $\mathrm{pre}(u)$ of some non-boundary vertex u assigned to some piece P (the following description applies both for micro and mini pieces), and denote with k the number of vertices of P. Note that any DFS must enter P via an entry vertex v, which is either the root or a boundary vertex. If the DFS leaves P via an exit vertex, the DFS can discover new vertices in P only if it enters via another entry vertex, or it backtracks over the exit vertex. By this we can observe two scenarios: vertex u is visited either (1) within the next k steps after the DFS enters via an entry vertex v, or (2) within the next k steps after a DFS backtracks over some exit vertex v. For (1) we have $|\mathrm{pre}(u) - \mathrm{pre}(v)| \leq k$ for some entry vertex v, and for (2) we have $|\mathrm{pre}(u) - \mathrm{post}(v)| \leq k$ for some exit vertex v. What we have shown is that the pre-order numbering of a non-boundary vertex u can be computed as a small *local offset* $k' \leq k$ plus the pre- or post-order numbering of some entry or exit vertex v, called the *reference boundary vertex*. It is crucial that the local offset is bounded by the number of vertices in P.

For a mini boundary vertex u, we store $\mathrm{pre}(u)$ and $\mathrm{post}(u)$ explicitly in an array with $\log n$ bits each. For a mini non-boundary vertex u, and its mini label u_i in a piece P_i, we store the local offset k' together with its reference boundary v (stored as its mini label v_i). Storing these values explicitly would use $O(n \log\log n)$ bits total, and as such we again employ a recursive strategy. We apply the mentioned strategy only for vertices being mini non-boundary vertices, that are also micro boundary vertices and we store the local offset and reference boundary explicitly. For non-boundary vertices $v_{i,j}$ of a micro piece $P_{i,j}$ we use the table lookup to compute both the local offset together with the reference boundary on-the-fly. This general strategy can be used for any strongly local value, possibly in combination with some additional structures required for more involved queries. Simple values such as the post-order, depth of a vertex, or the number of descendants, can be handled exactly the same way as the pre-order. For two of the more involved values, lowpoint and lowest-common ancestor, we describe the details next.

Lowpoints. Recall that $\mathrm{lowpoint}(u)$ of a vertex u is defined as the lowest pre-order number $\mathrm{pre}(v)$ such that v is reachable via a path that traverses the DFS subtree rooted at u together with at most one back edge of the palm tree. The lowpoint is a crucial value for typical DFS applications such as identifying biconnected components. We show that $\mathrm{lowpoint}(u)$ is strongly local. Assume that u is a non-boundary vertex in some piece P. Denote with k the number of vertices in P. First note that any back edge (u, v) must have v in P. Thus, $\mathrm{lowpoint}(u)$ either is $\mathrm{pre}(u)$, depends on a back edge that lies within P, or it depends on the lowpoint of a boundary vertex that is a descendant of u. Thus, we require the lowpoint of a boundary vertex of P, together with a local offset $k' \leq k$. See Figure 4a.

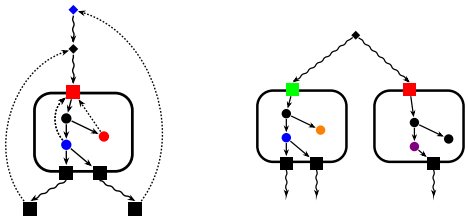


Figure 4 Lowpoints (a) The figure shows a piece and sketched parts of the DFS tree. Boundary vertices are drawn as squares, non-boundary vertices as circles, vertices of undefined type as diamonds. The lowpoint of the red non-boundary vertex is the red boundary vertex. The lowpoint of the blue non-boundary vertex is the blue diamond vertex. **LCAs (b)** The right figures shows two pieces. The LCA of the blue vertex and the violet vertex is the same as the LCA of the green and the red vertex. The LCA of the blue and the orange vertex is the green vertex. We can see that the LCA of two vertices in different pieces can be reduced to the LCA of two boundary vertices of the respective pieces.

The standard algorithm due to Hopcroft and Tarjan computes the lowpoints during a DFS traversal by updating it anytime the DFS backtracks to a vertex u and anytime a back edge (u, v) is discovered. For non-boundary vertices inside a micro piece, this update can be done via the table lookup using the augmentations provided by the lookup table for each boundary vertex of a micro piece $P_{i,j}$. Thus, anytime the DFS enters or leaves a micro piece $P_{i,j}$, we can update the current lowpoint values of each non-boundary vertex of $P_{i,j}$. As with the pre-order number, we store the required values for the boundary vertices explicitly in an array. The runtime of updating all values across the DFS is $O(n/\operatorname{poly}(\log\log n))$.

Lowest Common Ancestor. The operation $\operatorname{lca}(u, v)$ returns the root w of the smallest subtree T' of a tree T such that u and v are contained in T, i.e., it is the deepest vertex w in the tree such that u and v are descendants of w where deepest means it has the maximal distance from the root of T. We show that LCA queries for two vertices u, v can be reduced to strongly local values, one for each of the vertices. Let u, v be two non-boundary vertices of a piece P. If the LCA w of u, v lies within P, then the value is clearly local. If the LCA w does not lie within P, there must be a path from u to w and v to w such that each of these paths contains a boundary vertex u' and v', respectively. As there can be multiple such boundary vertices, let it be the first ones encountered by traversing the tree edges in reverse direction, i.e., the nearest such vertices. Then the LCA of u and v is equal to the LCA of u' and v'. Thus, for each pair of non-boundary vertices we only require to know the $\operatorname{lca}(u', v')$. The same is true, if u and v are non-boundary vertices in different pieces. Again, u' and v' are the nearest ancestors being boundary vertices in their respective pieces. Note that the choice of u' and v' is fixed for u and v, respectively, independent of the concrete query. See Figure 4b. To handle the LCA between mini boundary vertices, and for each mini piece, between micro boundary vertices, we use a known data structure to evaluate the LCA queries such as the data structure of Harel and Tarjan et al. [14]. For a tree with n' vertices, this data structure is initialized in $O(n')$ time and uses $O(n' \log n)$ bits such that LCA queries are available in linear time. We can not input our entire DFS tree to this data structure, and as such construct special smaller trees that capture the ancestry only between mini boundary vertices, and between micro boundary vertices for each mini piece. The time to construct the necessary structures is $O(n/\operatorname{poly}(\log\log n))$, and we use $o(n)$ bits.

We describe the concrete construction of the required smaller trees below. First observe the following. Let T' be some tree with vertex set V' and $S \subset V$ a set of *important vertices*, such that for pairs of vertices of S we want to know their LCAs. First, all leaves v in $V' \setminus S$ are irrelevant (can not be an LCA of a pair of vertices in S) and we recursively can remove them. Next consider vertices of $V' \setminus S$ of degree 2. These vertices are also irrelevant. We can replace such vertices v and its two edges $\{u, v\}\{v, w\}$ by a single edge $\{u, w\}$. This tree now contains the vertices of S and all vertices that are LCA's of pairs of vertices of S. Since we recursively removed all vertices of $V \setminus S$ with degree ≤ 2, the tree obtained has $O(|S|)$ vertices.

Now consider the DFS tree T and the set of mini boundary vertices. Taking the set of mini boundary vertices as the important vertices, we can see that we can shrink T such that it only contains $O(n/\operatorname{poly}(\log n))$ vertices without losing any information regarding the LCAs between boundary vertices. Analogously, for all micro boundary vertices inside a mini piece. The shrunken tree $T_{P_{i,j}}$ for a micro piece $P_{i,j}$ can be built via table lookup where we can support queries for every piece and every arbitrary set of important vertices S of the piece. Thus, we can obtain in $O(1)$ time a (set of) shrunken LCA tree(s) for piece $P_{i,j}$, which consists of the shrunken whole DFS tree with respect to the boundary vertices of the piece as important vertices. To build the shrunken tree for all mini boundary vertices, we iterate over all mini pieces $P_{i,j}$ and obtain shrunken $T_{P_{i,j}}$ with respect to the mini boundary vertices as important vertices, via table lookup. We so can build all required shrunken trees in $O(n/\operatorname{poly}(\log \log n))$ time using $o(n)$ bits.

▶ **Lemma 9.** *Let G be a separable graph given as a nested division $\mathcal{D}$ such that $\mathcal{D}$ encodes the palm tree of some DFS in $\mathcal{D}$. After $O(n/\operatorname{poly}(\log \log n))$ time preprocessing and using $o(n)$ bits we can provide the following queries.*

- children(u) *iteration over all children of u.*
- parent(u)*: output the parent of u.*
- pre(u)/post(u)*: return the pre-/post-order number of vertex u.*
- num-descendants(u)*: return the number of descendants of u.*
- depth(u)*: return the depth of u.*
- lowpoint(u)*: return the lowpoint of u.*
- lca(u, v)*: return the lowest-common ancestor of vertices u and v.*

Theorem 1 follows from first constructing the encoding of Lemma 6, and then constructing the palm tree of Lemma 8. Lemma 9 provides the queries. Next, we describe the details of constructing nested divisions efficiently.

3.3 Constructing Nested Divisions efficiently

For planar graphs we can construct the encoding of Lemma 6 efficiently, both in time and space. Kammer and Meintrup [21] presented a space-efficient algorithm for computing a $(\log n, r)$-relaxed divisions of minor closed graph classes that roughly works as follows. The input graph G is replaced with a minor F that contains $O(n/\log n)$ vertices such that each vertex of F represents $\Theta(\log n)$ vertices of G. Then, any algorithm $\mathcal{A}$ can be used for computing an $(1, r)$-relaxed division (a standard r-division) of F, which then can be turned into a $(\log n, r)$-relaxed division of G. As $\mathcal{A}$ is executed on a graph with $O(n/\log n)$ vertices, we achieve a speedup of a factor of $\Theta(\log n)$, and a space reduction by the same factor. Translation between the graph F and G uses linear time and bits. When recursively constructing divisions of mini pieces we can use standard algorithms as the graphs are small

enough. For planar graphs, linear time r-division algorithms [12, 23] exist, and thus the entire computation of the recursive division runs in linear time. For other graph classes the runtime depends on how efficiently separators can be found.

The final non-trivial part of constructing our encoding are related to the translation mappings between the different levels. The mapping is constructed using the powerful fully-indexable dictionary (FID) of Raman et al. [26] that, for a given universe $[\ell]$, uses $o(\ell)$ bits total when managing a set $S \subset [\ell]$ with $|S| = \Theta(\ell/\operatorname{poly}(\log \ell))$, which fits Blelloch and Farzan's use case. Effectively, this FID manages a compressed bit vector B of length ℓ where in B bits at index $i \in S$ are set to 1. Due to the use of Raman et al.'s FID their construction takes polynomial time $\omega(n)$ even for the simple case of encoding planar graphs and without regard to space-efficiency during the construction step. While Raman et al. mention their data structure can be constructed in expected linear time, a deterministic linear time construction is not known. This is due to the use of a certain hash function that is required. When succinctness is not required much simpler dictionaries suffices for the translation mappings. In this case, we effectively follow the same techniques outlined by Blelloch and Farzan, but use the indexable dictionary of Baumann and Hagerup, which can be constructed in $O(\ell/\log \ell)$ time and uses $\ell + o(\ell)$ bits for managing any set $S \subset [\ell]$:

▶ **Lemma 10** ([5]). *Let $B = (x_1, \ldots, x_\ell)$ be a bit string. In $O(\ell/\log \ell)$ time and $\ell + o(\ell)$ bits a data structure can be constructed such that afterward the following queries can be executed in constant time for $b \in \{0, 1\}$.*

- $\operatorname{rank}_b(i) = |\{j \in [i] | x_j = b\}|$
- $\operatorname{select}_b(i) = \min\{j \in [n] : \operatorname{rank}_B(j) = b\}$

Using the FID of Lemma 10 we are able to construct a compact variant of the encoding of Lemma 6 in $O(n)$ time and bits. If one is fine with expected linear time, the FID of Raman et al. [26] can be used and the encoding remains succinct. The functionalities of the different variants are the same.

▶ **Corollary 11** (Compact Nested Division). *For planar graphs, a compact (succinct) variant of the encoding of Lemma 6 can be constructed with $O(n)$ bits and $O(n)$ time (expected time).*

Combining Corollary 11 with Theorem 1 we are able to show Corollary 2. We give a more general version for arbitrary separable graphs next. In the following we denote with $f_{\mathcal{G}}(n)$ the runtime of an algorithm for graphs with n vertices of some separable graph class $\mathcal{G}$ that recursively computes a balanced separator until the graphs are small enough (of poly-logarithmic size), and with $f'_{\mathcal{G}}(n)$ the number of bits required by such an algorithm.

▶ **Corollary 12.** *Let G be a graph of a separable graph class $\mathcal{G}$. There exists a compact (succinct) encoding of G that can be constructed in $f_{\mathcal{G}}(n) + O(n)$ time (expected time) and $f'_{\mathcal{G}}(n) + O(n)$ bits that provides the same functionalities as the encoding of Theorem 1.*

References

1 Hüseyin Acan, Sankardeep Chakraborty, Seungbum Jo, and Srinivasa Rao Satti. Succinct data structures for families of interval graphs. In *16th International Symposium on Algorithms and Data Structures (WADS 2019)*, pages 1–13. Springer, 2019. doi:10.1007/978-3-030-24766-9_1.

2 Luca Castelli Aleardi, Olivier Devillers, and Gilles Schaeffer. Optimal succinct representations of planar maps. In *Proceedings of the Twenty-Second Annual Symposium on Computational Geometry (SCG 2006)*, pages 309–318. ACM, 2006. doi:10.1145/1137856.1137902.

3 Tetsuo Asano, Taisuke Izumi, Masashi Kiyomi, Matsuo Konagaya, Hirotaka Ono, Yota Otachi, Pascal Schweitzer, Jun Tarui, and Ryuhei Uehara. Depth-first search using $O(n)$ bits. In Hee-Kap Ahn and Chan-Su Shin, editors, *25h International Symposium on Algorithms and Computation (ISAAC 2014)*, pages 553–564. Springer, 2014. doi:10.1007/978-3-319-13075-0_44.

4 Niranka Banerjee, Sankardeep Chakraborty, Venkatesh Raman, and Srinivasa Rao Satti. Space efficient linear time algorithms for bfs, DFS and applications. *Theory Comput. Syst.*, 62(8):1736–1762, 2018. doi:10.1007/S00224-017-9841-2.

5 Tim Baumann and Torben Hagerup. Rank-select indices without tears. In *16th International Symposium on Algorithms and Data Structures (WADS 2019)*, pages 85–98. Springer, 2019. doi:10.1007/978-3-030-24766-9_7.

6 Guy E. Blelloch and Arash Farzan. Succinct representations of separable graphs. In *21st Annual Symposium on Combinatorial Pattern Matching (CPM 2010)*, pages 138–150. Springer, 2010. doi:10.1007/978-3-642-13509-5_13.

7 Sankardeep Chakraborty, Venkatesh Raman, and Srinivasa Rao Satti. Biconnectivity, Chain Decomposition and st-Numbering Using O(n) Bits. In *27th International Symposium on Algorithms and Computation (ISAAC 2016)*, volume 64 of *LIPIcs*, pages 22:1–22:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ISAAC.2016.22.

8 Yi-Ting Chiang, Ching-Chi Lin, and Hsueh-I Lu. Orderly spanning trees with applications to graph encoding and graph drawing. In *12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2001)*, pages 506–515. SIAM, 2001. URL: http://dl.acm.org/citation.cfm?id=365411.365518.

9 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 *LIPIcs*, pages 288–301. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.STACS.2015.288.

10 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.

11 Richard F. Geary, Rajeev Raman, and Venkatesh Raman. Succinct ordinal trees with level-ancestor queries. *ACM Trans. Algorithms*, 2(4):510–534, 2006. doi:10.1145/1198513.1198516.

12 M.T. Goodrich. Planar separators and parallel polygon triangulation. *Journal of Computer and System Sciences*, 51(3):374–389, 1995. doi:10.1006/jcss.1995.1076.

13 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.

14 Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984. doi:10.1137/0213024.

15 Jacob Holm, Giuseppe F. Italiano, Adam Karczmarz, Jakub Lacki, and Eva Rotenberg. Decremental SPQR-trees for Planar Graphs. In *26th Annual European Symposium on Algorithms (ESA 2018)*, volume 112 of *LIPIcs*, pages 46:1–46:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ESA.2018.46.

16 Jacob Holm and Eva Rotenberg. Good r-divisions imply optimal amortized decremental biconnectivity. *Theory Comput. Syst.*, 68(4):1014–1048, 2024. doi:10.1007/S00224-024-10181-Z.

17 John Hopcroft and Robert Tarjan. Algorithm 447: efficient algorithms for graph manipulation. *Commun. ACM*, 16(6):372–378, June 1973. doi:10.1145/362248.362272.

18 Taisuke Izumi and Yota Otachi. Sublinear-Space Lexicographic Depth-First Search for Bounded Treewidth Graphs and Planar Graphs. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, volume 168 of *LIPIcs*, pages 67:1–67:17, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2020.67.

19 G. Jacobson. Space-efficient static trees and graphs. In *30th Annual Symposium on Foundations of Computer Science (FOCS 1989)*, pages 549–554, 1989. doi:10.1109/SFCS.1989.63533.

20 Frank Kammer, Dieter Kratsch, and Moritz Laudahn. Space-efficient biconnected components and recognition of outerplanar graphs. *Algorithmica*, 81(3):1180–1204, 2019. doi:10.1007/S00453-018-0464-Z.

21 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 *LIPIcs*, pages 62:1–62:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ISAAC.2022.62.

22 Kenneth Keeler and Jeffery Westbrook. Short encodings of planar graphs and maps. *Discrete Appl. Math.*, 58(3):239–252, 1995. doi:10.1016/0166-218X(93)E0150-W.

23 Philip N. Klein, Shay Mozes, and Christian Sommer. Structured recursive separator decompositions for planar graphs in linear time. In *45th Annual ACM Symposium on Theory of Computing (STOC 2013)*, pages 505–514. Association for Computing Machinery, 2013. doi:10.1145/2488608.2488672.

24 Richard J. Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979. doi:10.1137/0136016.

25 J. Ian Munro, Patrick K. Nicholson, Louisa Seelbach Benkner, and Sebastian Wild. Hypersuccinct Trees - New Universal Tree Source Codes for Optimal Compressed Tree Data Structures and Range Minima. In *29th Annual European Symposium on Algorithms (ESA 2021)*, volume 204 of *LIPIcs*, pages 70:1–70:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ESA.2021.70.

26 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, 2007. doi:10.1145/1290672.1290680.

27 Omer Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4), September 2008. doi:10.1145/1391289.1391291.

28 Jens M. Schmidt. A simple test on 2-vertex- and 2-edge-connectivity. *Information Processing Letters*, 113(7):241–244, 2013. doi:10.1016/j.ipl.2013.01.016.

Incidences Between Curves and Points on the Grid

Esther Ezra
Department of Computer Science, Bar Ilan University, Ramat-Gan, Israel

Micha Sharir
School of Computer Science, Tel Aviv University, Israel

Abstract

We derive an improved upper bound for the number of incidences between the n vertices of a uniform grid and m convex or concave curves, each pair of which intersect in at most s points, for some integer parameter $s \geq 1$. For a square grid, our bound is

$$O\left(n^{2/3}m^{2/3} + m^{1-\frac{1}{3s}}n^{\frac{s+1}{3s}} + m + n\right).$$

This improves a general bound of $O(mn^{1/3})$ on the number of incidences with respect to vertices of a grid and convex or concave curves.

For a rectangular grid, which fits inside a $1 \times K$ rectangle, for some integer $K > 1$ (which generally may depend on n), the bound also depends on how large K is. The precise result is stated in Theorem 2, but, roughly, we get the same bound as above when K is not too large.

Our analysis competes with a celebrated result of Bombieri and Pila [6], which gives (usually) a sharper bound if we assume that the input curves are algebraic of constant degree and the input points are vertices of the square grid. However, the analysis in [6] strongly relies on these assumptions, and cannot be extended to handle the more general setup considered here.

As a main application, of independent interest, we present a variant of our technique for semi-algebraic range reporting on sets of points of "bounded spread" in the plane.

2012 ACM Subject Classification Theory of computation; Theory of computation → Computational geometry

Keywords and phrases Geometric incidences, uniform grid, bounded spread, Pick's theorem, range searching

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.30

Funding *Esther Ezra*: Work has been partially supported by Israel Science Foundation Grant 800/22 and Binational Science Foundation Grant 2022/131.
Micha Sharir: Work has been partially supported by Israel Science Foundation Grant 495/23.

1 Introduction

Let P be a set of n points in the plane, and let Γ be a set of m arbitrary curves from some fixed family of curves (e.g., lines, parabolas, circles, etc.). Let $I(P, \Gamma)$ denote the number of incidences between the points of P and the curves of Γ, that is, $I(P, \Gamma) = |\{(p, \gamma) | p \in P, \gamma \in \Gamma, p \in \gamma\}|$. We denote by $I(n, m)$ the maximum of $I(P, \Gamma)$, taken over all sets P of n points and sets Γ of m curves from the given family, implicit in this notation. The incidence problem is to bound $I(n, m)$. This problem has been studied extensively for more than four decades; see below.

Background. The simplest formulation of the incidence problem involves incidences between points and lines in the plane, where Szemerédi and Trotter [14] showed an upper bound of $O(n^{2/3}m^{2/3} + n + m)$, which is tight in the worst case [7] (see also [13, Chapter 1] and [11] for an overview of this and other results).

© Esther Ezra and Micha Sharir;
licensed under Creative Commons License CC-BY 4.0

For more general settings, the work of Sharir and Zahl [12] yields the following best known upper bound for $I(P,\Gamma)$, for arbitrary sets P of n points and Γ of m algebraic curves with t *degrees of freedom*, that is, the number of real parameters needed to specify a curve of Γ. The parameter t is also referred to as the *parametric dimension* of Γ. In this case the bound is

$$I(P,\Gamma) = O^*\Big(n^{2/3}m^{2/3} + n^{\frac{2t}{5t-4}}m^{\frac{5t-6}{5t-4}} + m + n\Big), \tag{1}$$

where the $O^*(\cdot)$ notation hides subpolynomial factors. See also the earlier works [3, 5, 10] for the case of curves with $t = 3$, such as circles or parabolas. Combined with the very recent result of Janzer et al. [8], these results yield the following slightly improved bound (with no subpolynomial factors):

$$I(P,\Gamma) = O\Big(n^{2/3}m^{2/3} + n^{6/11}m^{9/11} + m + n\Big). \tag{2}$$

An early celebrated work of Bombieri and Pila [6] has studied the case where the points of P form the vertex set of a $\sqrt{n} \times \sqrt{n}$ integer lattice (or grid, as we call it in this work), and each curve in Γ is an irreducible algebraic curve of degree d. They have shown, using a fairly involved algebraic analysis, that each such curve $\gamma \in \Gamma$ contains at most $O^*(n^{1/(2d)})$ lattice points, leading to an upper bound of $O^*(mn^{1/(2d)})$ on the number of incidences involving m such curves.

The analysis in [6] (as well as the more general case studied in [12] for arbitrary point sets) strongly relies on the algebraic structure of the input curves. If, however, the curves are not necessarily algebraic, but merely have a "well behaved" structure, then the analysis in [6] (and [12]) no longer applies in general. Specifically, we are interested in settings where the only assumptions are that each curve in Γ consists of $O(1)$ strictly convex or concave portions, and each pair of these curves intersect in at most s points, where $s > 0$ is a constant integer parameter. In such a case the best bound we are aware of is $O(mn^{1/3})$. This bound is mentioned in [13, Chapter 3], and, for the sake of completeness, we present below, in Section 3 (see Proposition 4), a full analysis that derives this bound, in a somewhat more general setup. The $O(mn^{1/3})$ bound is somewhat weak, since it does not depend on s, and it is also linear in m, so it tends to be large when $|\Gamma|$ is large. Our main goal in this paper is to obtain an improved incidence bound for arbitrary curves that satisfy the assumptions made above, and for points that lie on a square grid (as above), or, more generally, on a rectangular grid (which turns out to be a considerably more challenging problem).

Our results. In this work we significantly improve the "default" bound in Proposition 4 for the case where each pair of curves in Γ intersect in at most s points, for some constant integer parameter $s > 0$. We also allow the curves in Γ to contain line segments, and therefore weaken the assumption of strict convexity (or concavity). (This extension is essentially trivial, in view of the Szemerédi-Trotter bound [14].) We then obtain, in Section 2, the following bound for the case of a square grid:

▶ **Theorem 1.** *Let P be the set of the n vertices of the $\sqrt{n} \times \sqrt{n}$ grid, and let Γ be a collection of m curves, each of which consists of a constant number of convex and concave pieces, and every pair of which intersect in at most $s > 0$ points, for a constant integer parameter s. Then the number of incidences between P and Γ is*

$$I(P,\Gamma) = O\left(n^{2/3}m^{2/3} + n^{\frac{s+1}{3s}}m^{1-\frac{1}{3s}} + m + n\right).$$

In particular, one can easily verify that the bound in Theorem 1 is $O\left(n^{2/3}m^{2/3}+m+n\right)$ for pseudo-lines (curves with $s=1$), and $O\left(n^{2/3}m^{2/3}+n^{1/2}m^{5/6}+m+n\right)$ for pseudo-parabolas or pseudo-circle (curves with $s=2$), under the assumption that each curve can be decomposed into a constant number of convex/concave pieces. We comment that the bound in this theorem is worst-case tight for parabolas, using a construction inspired by Elekes (see, e.g., [13, Chapter 1]), but it requires a rectangular grid, which is the case we study in Section 3. In this latter section we assume that the points of P are the vertices of a $\sqrt{(n/K)}\times\sqrt{nK}$ uniform grid, for some parameter $1<K\leq n$ (this parameter is also referred to as the "aspect ratio" of P). For this setting, we present a more involved analysis, where we show:

▶ **Theorem 2.** *Let $1<K\leq n$ be an integer parameter (that may depend on n), and let P be the set of the n vertices of the $\sqrt{n/K}\times\sqrt{nK}$ uniform grid. Let Γ be a collection of m curves, each of which consists of a constant number of convex and concave pieces, and every pair of which intersect in at most $s>0$ points, for some constant integer parameter s. Then the number of incidences between P and Γ satisfies the following bounds.*

(i) *When $K\leq\left(\frac{n^{s+1}}{m}\right)^{\frac{1}{3s}}$, we have*

$$I(P,\Gamma)=O\left(n^{2/3}m^{2/3}+n^{\frac{s+1}{3s}}m^{1-\frac{1}{3s}}\log n+m+n\right).$$

(ii) *When $\left(\frac{n^{s+1}}{m}\right)^{\frac{1}{3s}}<K<n^{1/3}$ (which can only happen when $m\geq n$), we have*

$$I(P,\Gamma)=O(n^{2/3}m^{2/3}+n+Km\log n).$$

(iii) *When $K\geq n^{1/3}$, we have*

$$I(P,\Gamma)=O\left(n^{2/3}m^{2/3}+n+mn^{1/3}\right).$$

Note that case (i) in Theorem 2 essentially extends the bound in Theorem 1 to "nearly" square grids, whereas case (iii) implies that the bound is similar to the standard one (stated in Proposition 4), if the aspect ratio of the grid is relatively "large". The logarithmic factors in (i) and (ii) are artifacts of our analysis, and we believe that they can be removed with a more refined analysis.

Our proof technique is mainly combinatorial (i.e., avoids any algebraic considerations, which are irrelevant in our setup), and consists of two major ingredients: (i) A *Turán-type* bound [9] that is exploited as a black box (see Section 2 for more details), and (ii) a construction inspired by the "curvature-based" approach of Afshani and Cheng [1], who established an upper bound for the cost of semi-algebraic range reporting for planar semi-algebraic regions of constant complexity and points uniformly distributed in the unit square. We adapt, in a nontrivial manner, their machinery to derive incidence bounds for the case of point sets on a uniform grid, as specified in Theorems 1 and 2.

On the algorithmic front, following the analysis in [1], we present a variant of our solution for the square grid setting, for semi-algebraic range searching for points of "bounded spread" and planar semi-algebraic regions of constant complexity. Specifically, the input consists of a set P of n points, and the *spread* of P, defined as $\max_{p,q\in P}|pq|/\min_{p,q\in P}|pq|$, is $\Theta(\sqrt{n})$ (which is asymptotically the smallest possible). In particular, this assumption includes the case where each point of P lies close to a distinct grid vertex (of the $\sqrt{n}\times\sqrt{n}$ grid). The goal is to obtain a *space-query tradeoff bound for (online) semi-algebraic range reporting* for P, where the query ranges belong to a family $\mathcal{R}$ of semi-algebraic regions of parametric

dimension t (recall that t is the number of parameters needed to specify a range in $\mathcal{R}$). That is, for each query range, we want to efficiently report all the input points that lie in that range. In Section 4 we adapt the machinery in [1] and show:

▶ **Theorem 3.** *Let P be a set of n points in the plane with $\Theta(\sqrt{n})$ spread, and let $\mathcal{R}$ be a family of semi-algebraic ranges of parametric dimension t.*[1] *Then, for any value $\text{polylog}(n) \leq Q \leq c \cdot n^{1/4}$, where $c > 0$ is an appropriate absolute constant, we can construct a range reporting data structure for P, of size and preprocessing cost $O^*\left(n^2 + \frac{n^t}{Q^{3t-4}}\right)$, which supports range reporting queries with regions from $\mathcal{R}$, in time $O(Q + k)$ per query, where k is the output size.*

The main difference between our setting and the one presented in [1] is that in the latter the points in P are randomly and uniformly drawn in the unit square U, which makes the estimation of the expected number of points contained inside an region enclosed within U fairly straightforward. On the other hand, when we only assume that P has a $\Theta(\sqrt{n})$ spread, this estimation becomes trickier and requires further considerations.

2 The case of a square grid

Let P be the set of vertices of a $\sqrt{n} \times \sqrt{n}$ uniform grid that fits within the unit square $U = [0, 1] \times [0, 1]$ (assume, without loss of generality, that n is a square). Let Γ be a set of m arbitrary curves in the plane with the following properties:

- **(i)** Each curve $\gamma \in \Gamma$ does not contain any line segment.
- **(ii)** Any pair of curves $\gamma_1, \gamma_2 \in \Gamma$ intersect in at most s points, for some constant integer parameter $s > 0$. No pair of curves from Γ overlap.
- **(iii)** Each curve $\gamma \in \Gamma$ consists of $O(1)$ strictly convex and/or strictly concave portions.

For example, the case $s = 1$ corresponds to collections of *pseudo-lines* (that is, collections of unbounded Jordan curves, each pair of which intersect at most once), and the case $s = 2$ corresponds to collections of *pseudo-parabolas* (collections as above, where each pair intersect at most twice) or *pseudo-circles* (defined similarly to pseudo-parabolas, but the Jordan curves are closed), where we also require that these curves satisfy properties (i) and (iii).

In this section we derive an improved upper bound on the number $I(P, \Gamma)$ of incidences between the points of P and the curves of Γ. We comment that property (iii) is essential for the analysis, but property (i) can be discarded by applying the point-line incidence bound in [14] to the straight portions of the curves. Specifically, we can extend property (iii), so that each curve consists of at most $O(1)$ line segments (in addition to the $O(1)$ strictly convex/concave portions), so the total additional contribution to the incidence bound in this case is $O(n^{2/3}m^{2/3} + n + m)$. This bound can then be added to our bound on $I(P, \Gamma)$.

Our technique differs from the approach of Bombieri and Pila [6], who have studied the case where the curves of Γ are algebraic of constant degree, and derived a (generally) stronger bound for this case. Their work strongly relies on the algebraicity property of the curves. In contrast, our setting is more general and does not make such an assumption. Informally, our analysis is combinatorial, rather than algebraic, as in [6].

As described in Section 1, our analysis is based on the curvature-based approach of [1]. In this approach, a curve $\gamma \in \Gamma$ is covered by "canonical" slabs of varying widths. The fact that the total absolute curvature of γ is relatively small allows us to control (i) the thickness and

[1] We emphasize that the query ranges are not given in advance, but they are all defined by a fixed constant-complexity semi-algebraic predicate, with the t degrees of freedom as parameters.

number of slabs, and, consequently (ii) the number of points in a slab. These canonical slabs induce a decomposition on P, where thin slabs contain a relatively small number of points from P (see our analysis below for the specific bounds). Given this property, we apply a *Turán-type* incidence bound within each canonical slab, which eventually yields the incidence bound stated in Theorem 1 – see below.

The construction. Our analysis uses the following construction inspired by [1]. Let $1 \leq r \leq \sqrt{n}$ be an integer parameter, which we will set later. We partition the unit square U into a uniform $r \times r$ grid G. As a result, we obtain a partition of P into r^2 subsets, each confined to a grid cell τ of G, which has side length $1/r$ and contains $O(n/r^2)$ points of P. (Points on boundaries of cells of G are assigned arbitrarily to one of the neighboring cells, say, to either the lower or the left cell.) We then construct, for each cell $\tau \in G$, a family of *canonical slabs*, which we will use to cover the portions $\gamma \cap \tau$ of the curves $\gamma \in \Gamma$ that cross τ. The construction of these slabs proceeds as follows. For $i = 0, \ldots, \lfloor \log r \rfloor$, fix the parameter $\alpha_i = 2^i/r$ (which lies in $[0, 1]$). We first partition τ into vertical interior-disjoint slabs of width $\alpha_i/r = 2^i/r^2$ (and length $1/r$); the number of these slabs is $O(1/\alpha_i) = O(r/2^i)$. We then create $\lceil \pi r/2^i \rceil$ rotated copies of the i-th partition, so that the j-th copy is a rotation of the vertical partition by the angle $2^i j/r$. The slabs within each rotated family are extended or shrunk to fit within τ, so their lengths vary between 0 and $\sqrt{2}/r$. Let Σ_i be the collection of these canonical slabs, for a fixed i, over all cells $\tau \in G$ (and all rotations). We refer to the slabs in Σ_i as slabs of type i, or as i-slabs for short. We repeat the above construction for each $i = 0, \ldots, \lfloor \log r \rfloor$, and denote by Σ the union $\bigcup_i \Sigma_i$. Put $L_i = |\Sigma_i|$. By the above construction, L_i is the number of grid cells, which is r^2, times the number of i-slabs in a cell, which is $O((r/2^i) \cdot (1/\alpha_i)) = O(r^2/2^{2i})$; that is, we have $L_i = O(r^4/2^{2i})$.

Within each slab $\sigma \in \Sigma$ we bound the number of incidences between the points of $P \cap \sigma$ and a certain subfamily of the curves $\gamma \in \Gamma$ that intersect σ (see below for the precise definition). In order to do so, we need (i) to estimate $|P \cap \sigma|$, and (ii) to construct, for each $\gamma \in \Gamma$, a small-sized set of slabs in Σ that cover γ. Each slab σ interacts with all the curves that use σ for their cover – see below.

Bounding the number of points in a slab. Beginning with task (i), we apply *Pick's theorem* (see [4]), which measures the area of a nondegenerate simple polygon Q with vertices on the unit grid, in terms of the number of grid points that it contains. Specifically, let $\mu(Q)$ be the area of Q, let I (resp., B) be the number of points of P in the interior (resp., on the boundary) of Q. Then Pick's theorem asserts that $\mu(Q) = I + B/2 - 1$. In our case, each coordinate is shrunk by $\sqrt{n}$, so, for a nondegenerate simple polygon Q with vertices on P (which is the shrunk $\sqrt{n} \times \sqrt{n}$ grid), we have $n\mu(Q) = I + B/2 - 1$, implying that

$$|P \cap Q| = I + B \leq 2n\mu(Q) + 2.$$

Let $\sigma \in \Sigma$ be a canonical i-slab. Replace σ by the convex hull $C(\sigma) \subseteq \sigma$ of $P \cap \sigma$. [2] Then

$$|P \cap \sigma| = |P \cap C(\sigma)| \leq 2n\mu(C(\sigma)) + 2 \leq 2n\mu(\sigma) + 2 = O\left(n\alpha_i/r^2\right) = O\left(2^i n/r^3\right).$$

The additive term 2 is subsumed in this bound as long as r is not too large and i is not too small. When the bound $O\left(2^i n/r^3\right)$ is smaller than 1, we get $|P \cap \sigma| = O(1)$. Nevertheless, in this case too we will use the bound $O(2^i n/r^3)$, with some care, and explain later why this

[2] We apply this replacement since the vertices of σ do not necessarily belong to P, and this property is necessary in order to apply Pick's theorem.

does not cause any problems. We also note that this bound holds as long as $P \cap \sigma$ is not a collinear set of points (which is the case when $\mu(C(\sigma)) = 0$). Degenerate situations of this kind will be handled separately, in a rather simple manner – see below.

Covering a curve by canonical slabs. For task (ii), we follow a variant of the analysis in [1]. Let γ be a curve in Γ. We may assume that γ is (strictly) convex or (strictly) concave, and is both x-monotone and y-monotone; otherwise, by property (iii) we can partition γ into $O(1)$ pieces of this kind. The *turning angle* $\theta(\gamma')$ of a connected portion γ' of γ is the difference $|\theta_2 - \theta_1|$ between the orientations θ_1, θ_2 of the tangents to γ' at its endpoints. Because of convexity, the tangent orientations always increase or always decrease along γ', so the orientation of every tangent to γ', as well as that of the chord connecting the endpoints of γ', lies between θ_1 and θ_2. See Figure 1.

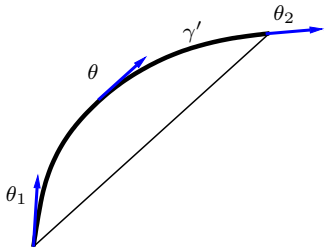


Figure 1 A concave curve γ' and the orientations θ_1, θ_2 at its endpoints, as well as the orientation θ of its chord.

Now let τ be a cell of G that γ crosses. By our assumptions, the intersection $\gamma' = \gamma \cap \tau$ is a single connected component of γ. Let θ' denote the turning angle of γ'. As is easily checked, γ' can be enclosed in a (not necessarily canonical) slab σ' of width $c\theta'/r$, for some absolute constant c. Let i be the index such that $2^{i-1}/r \leq \theta' < 2^i/r$. When $\theta' < 1/(2r)$ we still use the index 0 (see below), and we may assume that $\theta' \leq 1$; otherwise break γ' further into $O(1)$ pieces that satisfy this property.

Then, as is easily checked, σ' can be enclosed in the union of $O(1)$ canonical i-slabs. See also [1, Lemma 30] for a similar property. This is illustrated in Figure 2. (Note that γ' might end inside a cell, but this does not harm the analysis.)

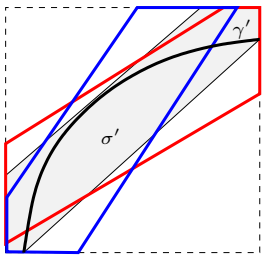


Figure 2 A curve γ' and the (non-canonical) slab σ' enclosing it (the shaded polygon) are covered by two canonical slabs (the red and blue polygons).

Fix an index i, and let Γ_i denote the collection of all the arcs $\gamma' = \gamma \cap \tau$, over the curves $\gamma \in \Gamma$ and the cells τ of G, for which $2^{i-1}/r \leq \theta(\gamma') < 2^i/r$. Since each piece in Γ_i consumes at least $2^{i-1}/r$ of the turning angle of the containing curve γ, the number of pieces of γ in Γ_i is at most $O(r/2^i)$. For pieces γ' with $\theta(\gamma') < 1/(2r)$ (which are left out of the preceding

analysis), the x-monotonicity itself implies that γ crosses at most $O(r)$ cells of G, allowing us to extend the first range to $[0, 1/r)$. That is, we have shown that each curve $\gamma \in \Gamma$ contributes $O(r/2^i)$ pieces to Γ_i, and each such piece can be covered by $O(1)$ (canonical) i-slabs.

Fix an i-slab σ, let $P(\sigma) = P \cap \sigma$, and $\Gamma_i(\sigma)$ be the set of the portions within σ of the curves of Γ that use σ in their cover (so their turning angles within the cell of G that contains σ lie in $[2^{i-1}/r, 2^i/r)$, or in $[0, 1/r)$ for $i = 0$). As shown earlier, assuming $P(\sigma)$ is not collinear,

$$n_\sigma := |P(\sigma)| = O(n\alpha_i/r^2) = O(2^i n/r^3). \tag{3}$$

Put $m_\sigma = |\Gamma_i(\sigma)|$, and observe that, summing over all i-slabs,

$$\sum_{\sigma \in \Sigma_i} m_\sigma = O(mr/2^i). \tag{4}$$

(This is because each covering slab "consumes" $\Omega(2^i/r)$ of the turning angle of the curve.) We now apply an incidence bound to $P(\sigma)$ and $\Gamma_i(\sigma)$, using the following considerations. Assume first that $P(\sigma)$ is not collinear. In such a case, we apply a moderate incidence bound using a *Turán-type* bound. Specifically, since each pair of curves in Γ intersect in at most s points, it follows that the incidence graph of $P(\sigma)$ and $\Gamma_i(\sigma)$ (and, in general, also the incidence graph of P and Γ) does not contain any copy of $K_{s+1,2}$. Therefore, by a result of Kövari, Sós and Turán [9], it follows that

$$I(P(\sigma), \Gamma_i(\sigma)) = O\left(n_\sigma m_\sigma^{1-1/(s+1)} + m_\sigma\right), \qquad \text{and} \tag{5}$$

$$I(P(\sigma), \Gamma_i(\sigma)) = O\left(m_\sigma n_\sigma^{1/2} + n_\sigma\right), \tag{6}$$

where the left-hand side is the number of edges in this incidence graph. In what follows, and unless stated otherwise, we will mostly be using the bound in (5).

If $P(\sigma)$ is collinear then, since we have assumed that Γ does not contain any line segments, each arc of $\Gamma_i(\sigma)$ can have at most two incidences with $P(\sigma)$ (since each such arc is strictly convex/concave, it must intersect any line in at most two points), for a total of at most $2m_\sigma$ incidences, which is subsumed in the above bound. In addition, if the bound in (3) on n_σ is smaller than 1, the actual value of n_σ is $O(1)$, in which case

$$I(P(\sigma), \Gamma_i(\sigma)) = O(|\Gamma_i(\sigma)|) = O(m_\sigma),$$

which is again subsumed in the above bound. Hence there is no harm if we use (with some care – see below) the bound in (3), even when it is smaller than 1. Letting Γ_i denote the collection $\bigcup_{\sigma \in \Sigma_i} \Gamma_i(\sigma)$, and summing this bound over all i-slabs σ, we thus obtain

$$I(P, \Gamma_i) = \sum_{\sigma \in \Sigma_i} I(P(\sigma), \Gamma_i(\sigma)) = O\Big(n_{\sigma,\max} \sum_{\sigma \in \Sigma_i} m_\sigma^{1-1/(s+1)} + \sum_{\sigma \in \Sigma_i} m_\sigma\Big),$$

where $n_{\sigma,\max}$ is the upper bound in (3). The following calculations handle slabs σ for which this bound is ≥ 1. Slabs for which this bound is smaller than 1 are handled in a much simpler manner – see below. Next, using Hölder's inequality, we obtain

$$I(P, \Gamma_i) = O\Big(n_{\sigma,\max} \sum_{\sigma \in \Sigma_i} m_\sigma^{1-\frac{1}{s+1}} + \sum_{\sigma \in \Sigma_i} m_\sigma\Big) \tag{7}$$

$$= O\Big(n_{\sigma,\max} \Big(\sum_{\sigma \in \Sigma_i} m_\sigma\Big)^{1-\frac{1}{s+1}} L_i^{\frac{1}{s+1}} + \sum_{\sigma \in \Sigma_i} m_\sigma\Big). \tag{8}$$

Recall that we have $\sum_{\sigma\in\Sigma_i} m_\sigma = O(mr/2^i)$, $n_{\sigma,\max} = O(2^i n/r^3)$ (where we assume for now that this bound is ≥ 1), and $L_i = O(r^4/2^{2i})$. Substituting these bounds in the above inequality, we obtain:

$$I(P,\Gamma_i) = O\left(\frac{2^i n}{r^3}\left(\frac{mr}{2^i}\right)^{1-1/(s+1)}\left(\frac{r^4}{2^{2i}}\right)^{1/(s+1)} + \frac{mr}{2^i}\right) = O\left(\frac{nm^{1-1/(s+1)}}{r^{2-3/(s+1)}2^{i/(s+1)}} + \frac{mr}{2^i}\right).$$

We sum these bounds over the $O(\log r)$ values of i for which $2^i \geq r^3/n$ (i.e., $n_{\sigma,\max} \geq 1$). Since i appears only in the denominators, we obtain decreasing geometric sums, which we can bound by their leading terms. We therefore obtain the incidence bound $O\left(\frac{nm^{1-1/(s+1)}}{r^{2-3/(s+1)}} + mr\right)$.

For i satisfying $2^i < r^3/n$ (note that there is no such i when $r \leq n^{1/3}$), the bound per slab is, as argued above, just $O(m_\sigma)$, and the sum of these bounds is just $O(mr/2^i)$, which, when summed over i, is just $O(mr)$. That is, we have $I(P,\Gamma) = O\left(\frac{nm^{1-1/(s+1)}}{r^{2-3/(s+1)}} + mr\right)$.

We balance these terms by choosing

$$r = \frac{n^{\frac{s+1}{3s}}}{m^{\frac{1}{3s}}} = \left(\frac{n^{s+1}}{m}\right)^{1/3s}.$$

For this choice to make sense, we need to require that $1 \leq r \leq n^{1/2}$. When r (in the above expression) is smaller than 1, we replace it by 1, and when $r > n^{1/2}$, we replace it by $n^{1/2}$. In the former case the first term decreases and the second term increases (and thus dominates the first term), so the bound becomes $O(m)$. In the latter case we have the relation $r > n^{1/2}$, which implies that $n^{s+1}/m > n^{3s/2}$, which is easily seen to be impossible for any $s \geq 2$ (unless m and n are constants). When $s = 1$ (the case of pseudo-lines), this inequality implies that m is at most $O(\sqrt{n})$. In this particular case, we apply the bound in (6) on the number of incidences, from which we obtain $I(P,\Gamma) = O\left(mn^{1/2} + n\right) = O(n)$. We have thus shown:

$$I(P,\Gamma) = O\left(m^{1-\frac{1}{3s}}n^{\frac{s+1}{3s}} + m + n\right). \tag{9}$$

Recall our discussion earlier in this section, where we also consider the case where property (i) does not hold for Γ, and, as a result, need to add $O(n^{2/3}m^{2/3} + n + m)$ to the bound. (When no line segments are allowed, we only get the bound in (9).) This completes the proof of Theorem 1.

3 The case of a rectangular grid

Problem statement. Let P be the set of n vertices of a uniform grid drawn within the rectangle $R = [0,1] \times [0,K]$, for some integer $1 < K \leq n$ (which may depend on n), and assume, without loss of generality, that $\sqrt{n/K}$ is an integer. We now construct a $\sqrt{n/K} \times \sqrt{nK}$ uniform grid within R, so each of its cells is a square of side length $\sqrt{K/n}$. Let Γ be a set of m curves satisfying properties (i)–(iii) of Section 2.

In this section we extend the analysis in the preceding section, to derive an improved upper bound on $I(P,\Gamma)$. Somewhat surprisingly, the analysis gets considerably more complicated in this case.

We begin with the following well-known property (see, e.g., [13]). For the sake of completeness, we present a proof that also holds for rectangular grids.

▶ **Proposition 4.**

(a) *An arbitrary strictly convex (or concave) curve can be incident to at most $O(n^{1/3})$ vertices of a square or rectangular uniform grid with a total of n vertices. The same bound holds for a curve that consists of $O(1)$ strictly convex and concave pieces.*

(b) *The bound is worst-case tight for a parabola in an $n^{1/3} \times n^{2/3}$ grid.*

Proof.

(a) Assume that the grid is a $k \times \ell$ grid, with $k\ell = n$. Without loss of generality, we may assume that the given curve γ is strictly convex and is both x- and y-monotone, and replace it by the polygonal convex curve γ_1 that connects the grid points on γ in a left-to-right order. An edge of γ_1 may contain many grid points, but we only consider the two extreme ones, because, by assumption, none of the interior points lies on γ. The vector along each edge of γ_1 is of the form $(\Delta x, \Delta y)$, where $\Delta x \in [1, k-1]$ and $\Delta y \in [1, \ell - 1]$. Fix two parameters u, v, whose values will be set later. An edge e of γ_1 is *long* if either $\Delta x \geq u$ or $\Delta y \geq v$, and *short* otherwise. Since each long edge advances by at least u columns to the right, or by at least v rows upwards, the number of long edges is at most $\frac{k}{u} + \frac{\ell}{v}$. For short edges, their slopes are of the form $\frac{\Delta y}{\Delta x}$, for $\Delta x \in [1, u-1]$, $\Delta y \in [1, v-1]$. The number of such slopes is at most uv, and because γ is strictly convex, the slopes of its edges are all distinct, implying that the number of short edges is at most uv.

We now set u and v to satisfy $\dfrac{k}{u} = \dfrac{\ell}{v} = uv$. The solution of these equations is $u = \dfrac{k^{2/3}}{\ell^{1/3}}$ and $v = \dfrac{\ell^{2/3}}{k^{1/3}}$. Hence the number of grid points on γ is $O(uv) = O\left(k^{1/3}\ell^{1/3}\right) = O\left(n^{1/3}\right)$. Note that this analysis holds when $k \geq \sqrt{\ell}$ and $\ell \geq \sqrt{k}$. When, say, $k < \sqrt{\ell}$, the claim is trivial, as the grid has at most $n^{1/3}$ columns. A symmetric argument applies when $\ell < \sqrt{k}$. If γ consists of $O(1)$ strictly convex or concave pieces, we apply the preceding analysis to each piece separately. This completes the proof of (a).

(b) The parabola $y = x^2$ passes through $n^{1/3}$ vertices of the $n^{1/3} \times n^{2/3}$ grid. ◀

The modified construction. We use the following modification of the earlier construction. While several parts of the analysis are similar, there are quite a few additional or different details that need to be worked out, to justify some overlap in the presentations. Let $1 \leq r \leq \sqrt{n/K}$ be an integer parameter, which we will set later.[3] We partition R into an $r \times rK$ uniform grid G, which yields a partition of P into r^2K subsets, each confined to a grid cell τ of G, each of which has side length $1/r$ and contains $O(n/(r^2K))$ points of P. As in Section 2, we then construct, for each cell $\tau \in G$, a family of canonical slabs, which we will use to cover the portions $\gamma \cap \tau$ of the curves $\gamma \in \Gamma$ that cross τ. The slabs are now of two kinds. The first family of slabs, which we refer to as $\Sigma^{(1)}$, is similar to the one constructed earlier for square grids, but with some nontrivial modifications (see below for details). The second family, denoted by $\Sigma^{(2)}$ (see below for the definition) is of a new, narrower kind of slabs, and most of this section analyzes the structural behavior of these slabs, and presents several combinatorial properties that we exploit in order to bound the number of incidences that these slabs contribute. Having these properties, the actual derivation of the incidence bound is mainly technical. The analysis for $\Sigma^{(1)}$ is similar, but simpler; see below.

[3] We will actually be using several different values of r; see below.

Generally speaking, the main difference between the setup here and the previous setup is that curves can cross a potentially larger number of cells of G, up to $\Theta(rK)$ of them, and this amount can potentially increase the incidence bound if we only use slabs of $\Sigma^{(1)}$ (informally, these slabs are too wide for this setup). To address this issue, we apply two main modifications: (i) We partition the curves in Γ into subarcs, depending on their "steepness" (see below), and bound the number of incidences with respect to each of these portions separately. (ii) We use a second family, namely $\Sigma^{(2)}$, of narrower slabs, which were not needed in the case of a square grid. We first introduce the structure of these curves and of the narrower slabs, and discuss the interaction between them, and then present the analysis for the total incidence bound.

Steepness and flatness. Consider a curve $\gamma \in \Gamma$ that crosses tr cells of G, for some $t > 2$; note that no such curves arose in the case of square grids (after having split each of them into monotone (strictly) convex/concave pieces). As in Section 2, we may assume that γ is (strictly) convex or concave and is both x-monotone and y-monotone; any curve in our family can be partitioned into $O(1)$ such curves. For concreteness, assume that γ is concave (i.e., its tangents turn clockwise as we go up), and is the graph of an increasing function; all the other cases, where γ is convex and/or decreasing, are treated in a fully symmetric manner. Now let τ be a cell of G that γ crosses. The intersection $\gamma' = \gamma \cap \tau$ is connected. We assign to γ' the orientations $\theta_1(\gamma') < \theta_2(\gamma')$, with respect to the y-axis, of the tangents to γ' at its bottom and top endpoints, respectively, and put $\Delta\theta(\gamma') = \theta_2(\gamma') - \theta_1(\gamma') > 0$.

The orientation of the chord that connects the endpoints of γ', as well as the orientations of all the tangents to γ', lie between $\theta_1(\gamma')$ and $\theta_2(\gamma')$. We define the *steepness* of γ' as the (larger) orientation $\theta_2(\gamma')$, and its *flatness* as $\Delta\theta(\gamma')$. Comparing to the analysis in Section 2, the flatness of a curve represents its turning angle (although we now consider much smaller values than those used in Section 2). The steepness is a new measure (that was not needed for the analysis in Section 2), whose role is to control the number of cells of G crossed by a curve (i.e., a subcurve with a given steepness).

We call γ' a (q, j)-*arc* if its steepness $\theta_2(\gamma')$ lies in $[1/2^j, 1/2^{j-1})$ and its flatness lies in $[1/(2^q r), 1/(2^{q-1} r))$, for suitable integer indices q and j. Only the case $q \geq 1$ is relevant; less flat curves can be handled by the wider slabs of $\Sigma^{(1)}$, and we treat them separately – see below. We assume that $j \leq \log K$, ignoring steeper arcs (because steeper arcs do not cross more cells, namely more than $O(rK)$ cells). The flatness parameter q, as just defined, determines the minimum width of a slab that can enclose γ', but nothing prevents us from using wider slabs, i.e., decrease the value of q. To emphasize the role of the minimum width, we denote the parameter q just defined as q^* (an intrinsic parameter of the arc), and refer to it as the *true flatness* parameter of the arc. See Figure 3 for an illustration.

Maximal j-steep subarcs. We extend the notions of steepness and flatness to larger portions of the original curve γ. Concretely, assume, as above and with no loss of generality, that γ is concave and monotone increasing. Let $\theta^- < \theta^+$ be the tangent orientations of γ (with respect to the y-axis) at its bottom and top endpoints, respectively. Let $j^- \geq j^+$ be the integers for which $\frac{1}{2^{j^-+1}} \leq \theta^- < \frac{1}{2^{j^-}}$ and $\frac{1}{2^{j^++1}} \leq \theta^+ < \frac{1}{2^{j^+}}$. We then partition γ into arcs $\gamma_{j^+}, \gamma_{j^++1}, \ldots, \gamma_{j^-}$, so that the tangent orientations of each γ_j lie in $[1/2^{j+1}, 1/2^j)$, and refer to γ_j as a *(maximal) j-steep* subarc (or portion) of γ. The flatness of γ_j is the sum of the flatness of its subarcs of intersections with the grid cells of G that it crosses. The definition of steepness is easily seen to imply that γ_j crosses $O(2^j r)$ cells of G. Indeed, if we consider the chord ℓ connecting the bottom and top endpoints of γ_j, we obtain a line segment whose

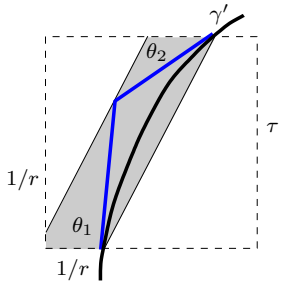


Figure 3 A (q^*, j)-arc γ' within a cell τ of G. Here $\theta_2 \approx \frac{1}{2^j}$, and $\theta_2 - \theta_1 \approx \frac{1}{2^{q^*}r}$. The width of the covering shaded slab is $\approx \frac{\theta_2-\theta_1}{r} = \frac{\Delta\theta(\gamma')}{r} \approx \frac{1}{2^{q^*}r^2}$.

slope (with respect to the y-axis) lies in $[1/2^{j+1}, 1/2^j)$, so the y-coordinate of the difference between the endpoints of γ_j is at most $O(2^j)$, and dividing by $1/r$ we get the asserted bound on the number of cells.

We now form[4] $O(\log K)$ families $\Gamma_0, \ldots, \Gamma_{\log K}$, where Γ_j consists of all maximal j-steep portions of curves from Γ, for $j = 0, \ldots, \log K$.[5] We slightly modify the definition of the first and last families Γ_0, $\Gamma_{\log K}$, so that Γ_0 consists of all j-steep portions whose tangent orientations lie in the range $[1, \infty)$, and for $\Gamma_{\log K}$ this range is $[0, 1/K)$. As will follow, this modification does not harm our analysis. An important property of this partition is that the subfamilies Γ_j are *independent of* r, which will be crucial for the forthcoming analysis. In what follows, we bound the number of incidences with respect to each family Γ_j separately, and then sum up these bounds over all such families. We comment that the bounds that we will obtain will depend on r, and optimizing each of them will require a different choice of r. Handling this multitude of values of r requires some careful treatment; see below.

Narrow slabs. Fix an index $j = 0, \ldots, \log K$, and consider a maximal j-steep subarc $\gamma \in \Gamma_j$. Let τ be a cell that γ crosses. Clearly, the intersection $\gamma' = \gamma \cap \tau$ is also j-steep (it may be even steeper). Let $\Delta\theta(\gamma')$ be the turning angle (flatness) of γ'. Again refer to Figure 3. If $\Delta\theta(\gamma') \geq 1/r$ we can (that is, should) use slabs from $\Sigma^{(1)}$ to cover γ'. Assume then that $\Delta\theta(\gamma') < 1/r$, and let $q^* \geq 0$ be the integer for which $1/(2^{q^*+1}r) \leq \Delta\theta(\gamma') < 1/(2^{q^*}r)$ (so γ' is a (q^*, j)-arc). Then we can cover γ' by a (non-canonical) slab σ of width at least $\Delta\theta(\gamma')/r = O(1/(2^{q^*}r^2))$ (see once again Figure 3). Note that we can use wider slabs to cover γ' (the actual width will be determined in the forthcoming analysis).[6] Let q be the integer parameter for the actual slab that we want to use (so its width is $\Theta(1/(2^q r^2))$, where $q \leq q^*$). To cover σ by $O(1)$ canonical slabs of this width, we need to use slabs rotated by multiples of the angle $1/(2^q r)$. But since γ is j-steep in τ, the orientations that we need to use (with respect to the y-axis) span an angular range of at most $O(1/2^j)$, so the total number of rotations is $O((1/2^j)/(1/(2^q r))) = O(2^{q-j}r)$. For each canonical orientation, the number of slabs at that orientation (within the current cell) is proportional to $1/r$ divided by the width, so it is $O(2^q r)$, for a total of $O(2^{2q-j}r^2)$ (q, j)-*slabs*, as we call them, in a cell, i.e., a total number of

$$L_{q,j} = O(2^{2q-j}r^4 K) \tag{10}$$

[4] For simplicity, and with no real effect on the analysis, we ignore here rounding issues.
[5] Note that this is slightly abusing the notation of Γ_i from Section 2, which has a different meaning.
[6] There is a tradeoff in using wider slabs: their number is smaller but each of them contains more points. The analysis below will address this issue.

(q, j)-slabs. We also refer to these slabs as q-slabs, when j is not important (not to be confused with i-slabs of $\Sigma^{(1)}$; see below and Section 2). We denote the family of these slabs as $\Sigma^{(2)}_{q,j}$, and denote their union, over all q, as $\Sigma^{(2)}_j$, and over all q and j, as $\Sigma^{(2)}$. We let $\Sigma = \Sigma^{(1)} \cup \Sigma^{(2)}$ denote the overall family of slabs.

▶ Remark 5. Note that it is crucial to apply the partition of γ into maximal j-steep portions, which eventually reduces the total number of rotations from $O(2^q r)$ to $O(2^{q-j} r)$. This yields a more refined bound on $L_{q,j}$, which we later exploit when deriving the overall incidence bound.

▶ Remark 6. Again, we emphasize that the analysis depends on the flexibility to use wider slabs to cover a (q^*, j)-arc, so we distinguish between the true flatness parameter q^* of an arc and the actual parameter $q \le q^*$ of the slabs that we use to cover the arc. In particular, the bound in (10) does not (explicitly) depend on the true flatness parameter q^*.

For technical reasons, we do not want to use too large a value for q (i.e., slabs that are too thin). Our actual choice is (recall that we must have $q \le q^*$)

$$q = \min\{2j, q^*\}. \tag{11}$$

Given a j-steep (portion of a) curve $\gamma \in \Gamma_j$, the number of cells τ from G that it crosses is $O(2^j r)$, as argued above. Alternatively, each (q^*, j)-arc consumes $\Theta(1/(2^{q^*} r))$ of the turning angle (flatness) of γ, so γ crosses at most $O((1/2^j)/(1/(2^{q^*} r))) = O(2^{q^*-j} r)$ cells of G (in which it behaves as a (q^*, j)-arc). Note that in this argument we have to use the true flatness parameter q^*. Hence the overall number of curve-slab interactions of this kind (that involve (q, j)-slabs and (q^*, j)-arcs) is

$$\sum_{\sigma \text{ is a } (q,j)\text{-slab}} m_\sigma = O\left(\min\{2^j,\ 2^{q^*-j}\} mr\right), \tag{12}$$

where m_σ is the number of (q^*, j)-arcs that use σ for their cover. See Figure 4.

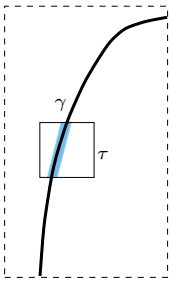


Figure 4 The portion of the curve γ inside the grid cell τ is a (q^*, j)-arc, which can be covered by $O(1)$ canonical (q, j)-slabs, for any $q \le q^*$; the figure illustrates one such slab by the shaded region.

Preparing for the incidence analysis. As before, within each slab $\sigma \in \Sigma^{(2)}$ we bound the number of incidences between the points of $P \cap \sigma$ and the subfamily of the j-steep curves $\gamma \in \Gamma_j$ (for each j separately) that use σ for the cover of their portion within the cell of σ. In order to do so, we need, as before, (i) to estimate $|P \cap \sigma|$, and (ii) to construct, for each $\gamma \in \Gamma_j$, a small-sized set $\Sigma^{(2)}(\gamma)$ of (canonical) slabs in $\Sigma^{(2)}$ that cover γ. Slabs in $\Sigma^{(1)}$ are handled in a simpler manner, similar to Section 2; see below. Task (ii) has already been addressed, so we mostly consider task (i), and briefly summarize (ii) afterwards.

Bounding the number of points in a slab. We apply Pick's theorem, as in Section 2, which now implies that, for a nondegenerate polygonal region Q whose vertices are grid points[7] $\frac{n}{K} \cdot \mu(Q) = I + B/2 - 1$, so, as long as $P \cap Q$ is not a collinear set of points,

$$|P \cap Q| = I + B \leq \frac{2n}{K}\mu(Q) + 2.$$

(i) Let $\sigma \in \Sigma^{(1)}$ be a canonical i-slab. Replace σ by the convex hull $C(\sigma) \subseteq \sigma$ of $P \cap \sigma$. We then have

$$|P \cap \sigma| = |P \cap C(\sigma)| \leq \frac{2n}{K}\mu(C(\sigma)) + 2 \leq \frac{2n}{K}\mu(\sigma) + 2 = O\left(\frac{n\alpha_i}{r^2 K}\right) = O\left(\frac{2^i n}{r^3 K}\right). \quad (13)$$

(ii) For a (q, j)-slab $\sigma \in \Sigma^{(2)}$, the same reasoning gives

$$|P \cap \sigma| = O\left(\frac{n}{K} \cdot \frac{1}{2^q r^3}\right) = O\left(\frac{n}{2^q r^3 K}\right). \quad (14)$$

Again, here q refers (naturally) to the thickness of the actual slab and not to the true flatness parameter q^* of any arc that uses the slab for its cover. These bounds hold as long as $P \cap \sigma$ is not collinear, and provided that they are at least 1. Degenerate situations, when one of these assumptions does not hold, are handled as in Section 2. That is, if $P \cap \sigma$ is collinear then σ contributes at most $2m_\sigma$ incidences, and the term $O(m_\sigma)$ in our bound takes care of this case too. Similarly, as in Section 2, when the bound on $|P \cap \sigma|$ in (13) or (14) is smaller than 1, we take the bound to be $O(1)$ instead. This does not violate the analysis (although some care is needed), as the bound that we derive subsumes the incident count in these cases – see below.

Covering a curve with canonical slabs. For task (ii), we just summarize the preceding analysis, which shows that the number of cells τ for which γ is j-steep and q^*-flat is $O(\min\{2^j, 2^{q^*-j}\}r)$. Hence, the overall number of curve-slab interactions, for (q, j)-slabs in $\Sigma_j^{(2)}$ and (q^*, j)-arcs, for $q \leq q^*$, is $O(\min\{2^j, 2^{q^*-j}\}mr)$.

Incidences within slabs of $\Sigma^{(1)}$. The analysis involving slabs of $\Sigma^{(1)}$ is similar to that in Section 2, although some steps of the analysis need to be adjusted to the rectangular setup. Here the width of a slab is associated with the index i (used in Section 2), which replaces the index q used above. The analysis that we present below is a refinement of the analysis in Section 2, where we consider the interaction of i-slabs with j-steep curves. This refinement was not needed in Section 2, but is needed here due to the fact that a curve $\gamma \in \Gamma$ may cross $\gg r$ cells of G, as mentioned above.

Let $\gamma \in \Gamma_j$ be a maximal j-steep curve, and let $\gamma' = \gamma \cap \tau$. We assume $\Delta\theta(\gamma') \geq 1/r$ (otherwise we use slabs of $\Sigma^{(2)}$ to cover γ), and let $i \geq 0$ be the integer for which $2^{i-1}/r \leq \Delta\theta(\gamma') < 2^i/r$. Analogously to the notation and definitions used above, we say that γ' is an (i, j)-*arc*. Then we can cover γ' by a (non-canonical) slab σ of width at least $\Delta\theta(\gamma')/r = O(2^i/r^2)$. As above, we can cover σ by $O(1)$ canonical slabs of this width, which are rotated by multiples of the angle $2^{i-1}/r$. But since γ is j-steep in τ, the orientations that we actually use (with respect to the y-axis) span an angular range of at most $O(1/2^j)$, so the total number of rotations in this case is $O((1/2^j)/(2^i/r)) = O(r/2^{i+j})$. For each canonical

[7] Here the scaling of the area is, as it should be, by n/K.

orientation, the number of slabs at that orientation (within the current cell) is proportional to $1/r$ divided by the width, so it is $O(r/2^i)$, for a total of $O(r^2/2^{2i+j})$ (i,j)-slabs in a cell (this is a refinement of the definition of i-slabs from Section 2), i.e., a total number of

$$L_{i,j} = O(r^4 K/2^{2i+j}) \tag{15}$$

(i,j)-slabs. We denote the family of these slabs as $\Sigma^{(1)}_{i,j}$, and denote their union, over all i, as $\Sigma^{(1)}_j$, and over all i and j, as $\Sigma^{(1)}$.

Fix an (i,j)-slab σ, let $P(\sigma) = P \cap \sigma$, and $\Gamma_{i,j}(\sigma)$ be the set of curves of Γ_j that use σ in their cover (so their turning angles within the cell of G that contains σ lie in $[2^{i-1}/r, 2^i/r)$). Let $\Gamma_{i,j}$ denote the collection, over all j-steep curves $\gamma \in \Gamma_j$ and (i,j)-slabs σ, of the portions $\gamma \cap \sigma$ that use σ for their cover. That is, $\Gamma_{i,j} = \bigcup_{\sigma \in \Sigma^{(1)}_{i,j}} \Gamma_{i,j}(\sigma)$. As shown earlier, when $P \cap \sigma$ is nondegenerate, we have in general

$$n_\sigma := |P(\sigma)| = O\left(\frac{n\alpha_i}{r^2 K}\right) = O\left(\frac{2^i n}{r^3 K}\right); \tag{16}$$

the cases where either $P \cap \sigma$ is collinear, or where the bound in (16) is smaller than 1, are handled as before, namely the incidence bound is $O(m_\sigma)$ in both cases, where $m_\sigma = |\Gamma_{i,j}(\sigma)|$. We next bound, for a j-steep curve $\gamma \in \Gamma_j$, the number of cells τ that it crosses. Each (i,j)-arc of γ consumes $\Theta(2^i/r)$ of the turning angle (flatness) of γ, and since γ is j-steep it crosses at most $O((1/2^j)/(2^i/r)) = O(r/2^{i+j})$ cells of G, in which it behaves as an (i,j)-arc. Hence the overall number of curve-slab interactions that involve (i,j)-slabs and (i,j)-arcs is[8]

$$\sum_{\sigma \text{ is an } (i,j)\text{-slab}} m_\sigma = O\left(\frac{mr}{2^{i+j}}\right). \tag{17}$$

We now apply the incidence bound (5) to $P(\sigma)$ and $\Gamma_{i,j}(\sigma)$, and obtain

$$I(P(\sigma), \Gamma_{i,j}(\sigma)) = O^*\Big(n_\sigma m_\sigma^{1-1/(s+1)} + m_\sigma\Big).$$

Recall that $\Gamma_{i,j} = \bigcup_{\sigma \in \Sigma^{(1)}_{i,j}} \Gamma_{i,j}(\sigma)$. By summing this bound over all (i,j)-slabs σ, we obtain

$$I(P, \Gamma_{i,j}) = \sum_{\sigma \in \Sigma^{(1)}_{i,j}} I(P(\sigma), \Gamma_{i,j}(\sigma)) = O\Big(n_{\sigma,\max} \sum_{\sigma \in \Sigma^{(1)}_{i,j}} m_\sigma^{1-1/(s+1)} + \sum_{\sigma \in \Sigma^{(1)}_{i,j}} m_\sigma\Big),$$

where $n_{\sigma,\max}$ is the upper bound in (16) (assuming it to be ≥ 1). Next, using Hölder's inequality, we obtain

$$\begin{aligned} I(P, \Gamma_{i,j}) &= O\Big(n_{\sigma,\max} \sum_{\sigma \in \Sigma^{(1)}_{i,j}} m_\sigma^{1-1/(s+1)} + \sum_{\sigma \in \Sigma^{(1)}_{i,j}} m_\sigma\Big) \\ &= O\Big(n_{\sigma,\max}\Big(\sum_{\sigma \in \Sigma^{(1)}_{i,j}} m_\sigma\Big)^{1-1/(s+1)} L_{i,j}^{1/(s+1)} + \sum_{\sigma \in \Sigma^{(1)}_{i,j}} m_\sigma\Big). \end{aligned}$$

Recall that we have $\sum_{\sigma \in \Sigma^{(1)}_{i,j}} m_\sigma = O(mr/2^{i+j})$, $n_{\sigma,\max} = \max\{O(2^i n/(r^3 K)), 1\}$ over $\sigma \in \Sigma^{(1)}_{i,j}$, and $L_{i,j} = O(r^4 K/2^{2i+j})$. Substituting these bounds in the above inequality, we obtain

[8] Here there is no need to use wider slabs, as in the case of $\Sigma^{(2)}$.

$$I(P,\Gamma_{i,j}) = O\left(\frac{2^i n}{r^3 K}\left(\frac{mr}{2^{i+j}}\right)^{1-1/(s+1)}\left(\frac{r^4 K}{2^{2i+j}}\right)^{1/(s+1)} + \frac{mr}{2^{i+j}}\right)$$
$$= O\left(\frac{nm^{1-1/(s+1)}}{K^{s/(s+1)}r^{2-3/(s+1)}2^{i/(s+1)}2^j} + \frac{mr}{2^i 2^j}\right).$$

We sum these bounds over the $O(\log r)$ values of i. Since i appears only in the denominators, we obtain decreasing geometric sums, which we can bound by their leading terms, in which $2^i = 1$. Denoting by $I_1(P,\Gamma_j)$ the number of incidences within slabs of $\Sigma_j^{(1)}$, we therefore obtain

$$I_1(P,\Gamma_j) = O\left(\frac{nm^{1-1/(s+1)}}{K^{s/(s+1)}r^{2-3/(s+1)}2^j} + \frac{mr}{2^j}\right). \tag{18}$$

Incidences within slabs of $\Sigma^{(2)}$. Consider next incidences within slabs $\sigma \in \Sigma_j^{(2)}$, for a fixed index j. Let $\gamma \in \Gamma_j$ be a j-steep curve for which we need to use slabs of $\Sigma_j^{(2)}$. To simplify the analysis, and with no loss of generality, assume that (i) γ is concave and monotone increasing, and (ii) γ enters G through its bottom edge and leaves G through its right or top edge, after having crossed tr cells of G, for some $2 < t \le K+1$.

Let τ be a cell of G that γ crosses. We use the notations introduced earlier concerning the steepness and flatness of $\gamma' = \gamma \cap \tau$, and let j and q^* be the parameters for which γ' is a (q^*,j)-arc. (As already noted, γ' may be steeper within τ, but we will, as we can, use the same steepness index j for γ' too.) As argued, γ' can be enclosed by $O(1)$ canonical (q,j)-slabs, for any $q \le q^*$.

Recalling the bounds in (10), (12), and (14), the overall number of (q,j)-slabs is $L_{q,j} = O(2^{2q-j}r^4K)$, the number of points in a q-slab is $n_\sigma = O\left(\frac{n}{2^q r^3 K}\right)$ (assuming that this bound is ≥ 1), and

$$\sum_{\substack{\sigma \text{ is a slab}\\ \text{covering a } (q^*,j)\text{-arc}}} m_\sigma = O\left(\min\{2^j,\ 2^{q^*-j}\}mr\right).$$

Let $\Gamma_{q^*,j}(\sigma)$ denote the set of (q^*,j)-arcs that use σ for their cover (so $m_\sigma = |\Gamma_{q^*,j}(\sigma)|$). Again, we may assume that $P(\sigma)$ is not a collinear set of points, otherwise, the number of incidences is at most $2m_\sigma$, as argued in Section 2. We also recall that the bound $O(m_\sigma)$ takes care of incidences within slabs σ for which the bound on n_σ is smaller than 1. Thus, we can now apply the incidence bound (5) to $P(\sigma)$ and $\Gamma_{q^*,j}(\sigma)$, and obtain

$$I(P(\sigma),\Gamma_{q^*,j}(\sigma)) = O\Big(n_\sigma m_\sigma^{1-1/(s+1)} + m_\sigma\Big).$$

Hence, fixing q^*, denoting by $\Gamma_{q^*,j}$ the collection of all (q^*,j)-arcs, and summing the above bound over the set $\Sigma^{(2)}_{q,q^*,j}$ of all (q,j)-slabs σ that are used to cover (q^*,j)-arcs, for any (fixed) $q \le q^*$, we obtain:

$$I_q(P,\Gamma_{q^*,j}) = \sum_{\sigma\in\Sigma^{(2)}_{q,q^*,j}} I(P(\sigma),\Gamma_{q^*,j}(\sigma)) = O\Big(n_{\sigma,\max}\sum_{\sigma\in\Sigma^{(2)}_{q,q^*,j}} m_\sigma^{1-1/(s+1)} + \sum_{\sigma\in\Sigma^{(2)}_{q,q^*,j}} m_\sigma\Big),$$

where I_q refers to incidences within q-slabs, and where, as before, in the first sum we replace the uniformly bounded quantities n_σ by their upper bound $n_{\sigma,\max} = O(n/(2^q r^3 K))$

(assuming that this expression is ≥ 1), over all $\sigma \in \Sigma^{(2)}_{q,q^*,j}$. Next, using Hölder's inequality, we obtain

$$\begin{aligned} I_q(P,\Gamma_{q^*,j}) &= O\Big(n_{\sigma,\max} \sum_{\sigma\in\Sigma^{(2)}_{q,q^*,j}} m_\sigma^{1-1/(s+1)} + \sum_{\sigma\in\Sigma^{(2)}_{q,q^*,j}} m_\sigma\Big) \\ &= O\Big(n_{\sigma,\max}\Big(\sum_{\sigma\in\Sigma^{(2)}_{q,q^*,j}} m_\sigma\Big)^{1-1/(s+1)} L_{q,j}^{1/(s+1)} + \sum_{\sigma\in\Sigma^{(2)}_{q,q^*,j}} m_\sigma\Big). \end{aligned} \tag{19}$$

We substitute $\sum_{\sigma\in\Sigma^{(2)}_{q,q^*,j}} m_\sigma = O\left(\min\{2^j,\ 2^{q^*-j}\}mr\right)$, $n_{\sigma,\max} = O(n/(2^q r^3 K))$ (assuming this bound to be $\Omega(1)$), and $L_{q,j} = O(2^{2q-j}r^4K)$ in (19). We bifurcate according to whether $q^* \geq 2j$, and then write 2^j for $\min\{2^j,\ 2^{q^*-j}\}$, or $q^* \leq 2j$, and then write 2^{q^*-j} for that expression.

When $q^* \geq 2j$ we obtain

$$\begin{aligned} I_q(P,\Gamma_{q^*,j}) &= O\Big(n/(2^q r^3 K)\left(2^j mr\right)^{1-1/(s+1)}\left(2^{2q-j}r^4K\right)^{1/(s+1)} + 2^j mr\Big) \\ &= O\left(\frac{nm^{1-1/(s+1)}}{2^{q-j-\frac{2q-2j}{s+1}}K^{1-1/(s+1)}r^{2-3/(s+1)}} + 2^j mr\right). \end{aligned} \tag{20}$$

When $q^* \leq 2j$ we obtain

$$\begin{aligned} I_q(P,\Gamma_{q^*,j}) &= O\Big(n/(2^q r^3 K)\left(2^{q^*-j}mr\right)^{1-1/(s+1)}\left(2^{2q-j}r^4K\right)^{1/(s+1)} + 2^{q^*-j}mr\Big) \\ &= O\left(\frac{nm^{1-1/(s+1)}}{2^{j+q-2q/(s+1)-q^*s/(s+1)}K^{1-1/(s+1)}r^{2-3/(s+1)}} + 2^{q^*-j}mr\right). \end{aligned} \tag{21}$$

In the former case, when $q^* \geq 2j$ we take $q = 2j$, so the bound in (20) becomes

$$O\left(\frac{nm^{1-1/(s+1)}}{2^{j-2j/(s+1)}K^{1-1/(s+1)}r^{2-3/(s+1)}} + 2^j mr\right).$$

In the latter case, when $q^* \leq 2j$ we take $q = q^*$, observe that $q^* - j \leq j$, and conclude that the bound in (21) becomes

$$O\left(\frac{nm^{1-1/(s+1)}}{2^{j-q^*/(s+1)}K^{1-1/(s+1)}r^{2-3/(s+1)}} + 2^j mr\right) = O\left(\frac{nm^{1-1/(s+1)}}{2^{j-2j/(s+1)}K^{1-1/(s+1)}r^{2-3/(s+1)}} + 2^j mr\right).$$

That is, both expressions in (20) and (21) can be bounded by

$$I_q(P,\Gamma_{q^*,j}) = O\left(\frac{nm^{1-1/(s+1)}}{2^{j-2j/(s+1)}K^{1-1/(s+1)}r^{2-3/(s+1)}} + 2^j mr\right). \tag{22}$$

(Note that this bound is independent of q^*.) It is easy to verify that the bound in (22) dominates the bound (18) for the number of incidences within slabs of $\Sigma^{(1)}_j$ (for the same j). So we now continue the analysis using the bound (22).

We next choose r to balance the two terms. That is, for j fixed,

$$r = r_j = \frac{n^{\frac{s+1}{3s}}}{m^{\frac{1}{3s}}(2^{2j}K)^{1/3}}. \tag{23}$$

Before continuing, we note that we have $O(\log K)$ different values of $r = r_j$, which implies that we need to construct a different grid G for every family Γ_j. This, however, does not harm the analysis, since, as already noted, the families Γ_j are independent of r.

For each $j = 0, \ldots, \log K$, for the choice of $r = r_j$ in (23) to make sense, we require that $1 \leq r_j \leq \sqrt{n/K}$. When $r_j < 1$ we replace it by 1, and when $r_j > \sqrt{n/K}$ we replace it by $\sqrt{n/K}$. Since the values of r_j form a geometric decreasing sequence, the requirement $1 \leq r_j \leq \sqrt{n/K}$ for each $j = 0, \ldots, \log K$ holds iff $r_{\log K} \geq 1$ and $r_0 \leq \sqrt{n/K}$. So it suffices to consider the extreme cases $r_{\log K} < 1$ and $r_0 > \sqrt{n/K}$. Consider first the case $r = r_{\log K} < 1$. This constraint is:

$$\frac{1}{K}\left(\frac{n^{s+1}}{m}\right)^{\frac{1}{3s}} < 1, \quad \text{or} \quad K > \left(\frac{n^{s+1}}{m}\right)^{\frac{1}{3s}}. \tag{24}$$

Replacing r by 1 makes the second term in (22) dominate the first term. Hence, the incidence bound becomes $I_q(P, \Gamma_{q^*,j}) = O(2^j m)$ in this case. Then, summing up over all $O(\log^2 K)$ pairs (q, j) (recall that $j \leq \log K$ and $q \leq 2j$), we obtain a total incidence bound of $O(Km \log K)$. We comment, however, that when K exceeds $n^{1/3}$, we use instead the general bound in Proposition 4, which implies that the number of incidences is $O(mn^{1/3})$. (The lower bound in (24) exceeds $n^{1/3}$ when $m \leq n$.)

Suppose next that $r = r_0$ exceeds $\sqrt{n/K}$. This implies that

$$K^{1/6} > m^{\frac{1}{3s}} n^{\frac{1}{2} - \frac{s+1}{3s}} = m^{\frac{1}{3s}} n^{\frac{s-2}{6s}}, \qquad \text{or} \qquad K > m^{2/s} n^{(s-2)/s}.$$

Since we only consider the case $K \leq n^{1/3}$, this implies that $m < n^{1-s/3}$, which is impossible when $s \geq 3$ (unless m and n are constants), and becomes $m < n^{1/3}$ when $s = 2$, and $m < n^{2/3}$ when $s = 1$. In these two latter cases we apply the general bound $O(mn^{1/3})$ in Proposition 4, which in these cases is at most $O(n)$. Note that these are global bounds that do not depend on the type of the slabs in ($\Sigma^{(1)}$ or $\Sigma^{(2)}$).

In the remaining range, i.e., for $1 \leq r = r_j \leq \sqrt{n/K}$, we reason as follows. Due to the choice in (23), and using (22), the bound simply becomes

$$I_q(P, \Gamma_{q^*,j}) = O(2^j mr) = O\left(\frac{2^j \left(n^{(s+1)/(3s)} m^{1-1/(3s)}\right)}{(2^{2j}K)^{1/3}}\right) = O\left(\frac{\left(2^{j/3} n^{(s+1)/(3s)} m^{1-1/(3s)}\right)}{K^{1/3}}\right).$$

We next sum up this bound over all $O(\log^2 K)$ pairs (q, j), and obtain (recall that the actual sum is over all (q, j)-slabs σ in $\Sigma^{(2)}_{q,q^*,j}$, and thus the actual summation does not depend explicitly on q^*):

$$I_2(P, \Gamma) = O\left(\sum_{q,j} \frac{\left(2^{j/3} n^{(s+1)/(3s)} m^{1-1/(3s)}\right)}{K^{1/3}}\right) = O\left(n^{(s+1)/(3s)} m^{1-1/(3s)} \log K\right), \tag{25}$$

where $I_2(P, \Gamma)$ is the overall number of incidences within slabs of $\Sigma^{(2)}$.

We also recall that we need to add the bound $O(n^{2/3}m^{2/3} + n + m)$, for the case where the curves in Γ contain line-segments (see above). The proof of Theorem 2 now follows by combining all the above bounds, including the bounds derived for $\Sigma^{(1)}$, and using the fact that $\log K = O(\log n)$.

4 Range Reporting for Point Sets of Bounded Spread

In this section we prove Theorem 3. Our construction is based on that of Afshani and Cheng [1], who obtained a similar space-query tradeoff bound for (online) semi-algebraic range reporting for points uniformly distributed in the unit square. Here we consider instead the case where the input set of n points has spread $\Theta(\sqrt{n})$. This setup requires some extra care, as we provide below.

The grid construction. Let P be a set of n points in the plane of $\Theta(\sqrt{n})$ *spread*, that is, the ratio between the maximum and the minimum distances among pairs of points from P is at most $\Theta(\sqrt{n})$. (As already noted, this is asymptotically the smallest possible value of the spread of n points in the plane.) Suppose, without loss of generality, that all the points of P lie in the unit square $U = [0,1] \times [0,1]$. We now draw a $\sqrt{n} \times \sqrt{n}$ uniform grid G_0 that fits within U; let us denote by V the set of its vertices. Since the spread of P is $\Theta(\sqrt{n})$, each cell of G_0 contains only $O(1)$ points of P (or is empty), as is easily verified.

We next apply a similar grid construction as in Section 2, where now the parameter r is the query time $Q = Q(n)$. For simplicity of presentation, we continue to denote it by r. We partition the unit square U into an $r \times r$ uniform grid G. As a result, we obtain a partition of V into r^2 subsets, each confined to a grid cell τ of G, which has side length $1/r$ and contains $O(n/r^2)$ points of V. Due to the bounded spread property, this also forms a partition of P such that $|P \cap \tau| = O(n/r^2)$, for each cell τ of G. (Points on boundaries of cells of G are assigned arbitrarily to one of the neighboring cells, as before.) We then construct, for each cell $\tau \in G$, a family of canonical slabs in an identical manner to that presented in Section 2. These slabs are used to cover portions of the boundaries of query ranges R. We use the same notation as in Section 2, and denote by Σ_i the set of all i-slabs, for each $i = 0, \ldots, \lfloor \log r \rfloor$, and put $\Sigma = \bigcup_i \Sigma_i$. We recall that $L_i = |\Sigma_i| = O(r^4/2^{2i})$.

In each slab $\sigma \in \Sigma$ we construct a range-reporting data structure for the set $P \cap \sigma$ (see below for details). In what follows we first estimate $|P \cap \sigma|$, and then present the construction of the range-reporting data structure.

Bounding $|P \cap \sigma|$. We first estimate $|V \cap \sigma|$ and then show how to obtain a similar bound for $|P \cap \sigma|$. We apply *Pick's theorem* as in Section 2, and use the notation there. That is, for a nondegenerate simple polygon Q with vertices of G_0, we have $n\mu(Q) = I + B/2 - 1$, implying that $|V \cap Q| = I + B \leq 2n\mu(Q) + 2$. Then for a canonical i-slab $\sigma \in \Sigma$, we obtain (where $C(\sigma) \subseteq \sigma$ is the convex hull of $V \cap \sigma$):

$$|V \cap \sigma| = |V \cap C(\sigma)| \leq 2n\mu(C(\sigma)) + 2 \leq 2n\mu(\sigma) + 2 = O\left(n\alpha_i/r^2\right) = O\left(2^i n/r^3\right).$$

As already noted, this bound holds as long as $V \cap \sigma$ is not a collinear set of points, and the additive term 2 is subsumed in this bound as long as r is not too large. We next present an appropriate choice for r, such that both of these properties hold:

▶ **Lemma 7.** *If $r \leq cn^{1/4}$, for a sufficiently small constant $c > 0$, then, for each canonical slab $\sigma \in \Sigma$, the set $V \cap \sigma$ is not collinear.*

Proof. Let τ be the grid cell that contains σ. By the condition $r \leq cn^{1/4}$, the width of σ must be at least $1/r^2 = \Omega(1/\sqrt{n})$. If c is sufficiently small, then σ must contain in its interior a grid square of G_0 (which is an axis-parallel $(1/\sqrt{n}) \times (1/\sqrt{n})$ square, whose vertices belong to V). Therefore, σ contains four points of V, which are not at a collinear position. ◀

We thus conclude that we always have $|V \cap \sigma| = O\left(2^i n/r^3\right)$, using the above choice of r; for this choice, the term $2^i n/r^3$ is at least $n^{1/4}$, which, in particular, dominates the additive term 2. Using the notation of Lemma 7, it is also easy to verify that each grid square of G_0 intersected by σ must have at least one vertex $v \in V$ that lies inside σ. In other words, the number of grid squares intersected by σ is equal, up to a constant factor, to the number of grid points inside σ. Since each grid square contains $O(1)$ points of P, this also asymptotically bounds $|P \cap \sigma|$. In other words, we showed (for r as above):

$$|P \cap \sigma| = O\left(2^i n/r^3\right), \quad \text{for each } i\text{-slab } \sigma. \tag{26}$$

Since we rely on Lemma 7, we require that $Q(n) = r \leq cn^{1/4}$.

The data structure. Let P and $\mathcal{R}$ be as in the statement of Theorem 3. We follow the analysis of [1] and construct a two-part data structure for P that supports range reporting queries from $\mathcal{R}$, as follows. Let $R \in \mathcal{R}$ be a query range. At the first structure we efficiently report points that are close to ∂R, and at the second structure we use a simplex range reporting data structure with space complexity $O^*(n^2)$ and $O(\log n + k)$ query time, where k is the output size. For the first task we use the above grid construction, where the parameter r is the total query time-bound parameter $Q(n)$ that we aim to achieve.

As observed in [1], after covering ∂R by canonical slabs, the leftover (inner) portion R° of R is a polygonal region, whose complexity is proportional to the number of slabs used to cover ∂R. Thus R° can be covered by triangles, whose number is proportional to that quantity. Specifically, we have $O(1)$ triangles in each grid cell τ of G crossed by ∂R, for a total of $O(r) = O(Q(n))$ such triangles. The cells that are fully contained in R° are easier to handle, since all the points of P in these cells belong to the output; we refer to [1] for the missing details. Hence, using $O(Q(n))$ simplex range reporting queries, we can report all the points of P in R°. That is, the cost of this part is (as argued in [1]) $O(Q(n)\log n + k_2)$ time, where k_2 is the number of points from P in R°. (Recall that we use $O^*(n^2)$ space in this structure).

We thus focus on range reporting near the boundary of R. Let us consider an i-slab $\sigma \in \Sigma_i$. Recall from the above discussion (and Section 2) that the corresponding parameter α_i is $2^i/Q(n)$, $L_i = Q(n)^4/2^{2i}$, and $n_\sigma = |P \cap \sigma| = O(n\alpha_i/Q(n)^2) = O(2^i n/Q(n)^3)$ (see (26)). We now construct within σ a range-reporting data structure, with query time $O(2^i + k_1)$ (where k_1 is the output size) and space complexity $O\left(\left(\frac{n_\sigma}{2^i}\right)^t\right)$. Such a data structure is fairly standard and is obtained by duality, see [1] and the references therein. As argued in Section 2, ∂R uses at most $O(1/\alpha_i) = O(Q(n)/2^i)$ i-slabs σ for its cover, so the total query time overhead of this part is $O(Q(n))$; we refer to [1, Lemma 17] for further details concerning this property.

We thus conclude that the total space complexity is

$$O^*\left(\sum_{i=0}^{\lfloor \log Q(n)\rfloor} \sum_{\sigma\in\Sigma_i} \left(\frac{n_\sigma}{2^i}\right)^t\right) = O^*\left(\sum_{i=0}^{\lfloor \log Q(n)\rfloor} Q(n)^4/2^{2i}\left(\frac{2^i n\alpha_i/Q(n)^3}{2^i}\right)^t\right) = O^*\left(\frac{n^t}{Q(n)^{3t-4}}\right).$$

Recall that due to the simplex range reporting data structure stored at our second structure, we need to add a factor of $O^*(n^2)$ to the storage (and preprocessing) complexity. The threshold for $Q(n)$ in Theorem 3 follows from the threshold in Lemma 7. We also note that we require $Q(n) \geq \text{polylog}(n)$, as follows from standard properties of range search (see [1, 2]).

We comment that due to the second data structure, where we use simplex range reporting, the actual query time is $O(Q(n)\log n + k)$, however, we can easily reduce it to $O(Q(n) + k)$ by choosing $r = Q(n)/\log n$ (and keeping the storage complexity $O^*(n^2)$), as is easily verified. This completes the proof of Theorem 3.

References

1 P. Afshani and P. Cheng. On semialgebraic range reporting. 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 3:1–3:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.SOCG.2022.3.

2 P. K. Agarwal. Simplex range searching. In M. Loebl, J. Nešetřil, and R. Thomas, editors, *Journey Through Discrete Mathematics*, pages 1–30. Springer Verlag, Heidelberg, 2017.

3 P. K. Agarwal, E. Nevo, J. Pach, R. Pinchasi, M. Sharir, and S. Smorodinsky. Lenses in arrangements of pseudo-circles and their applications. *J. ACM*, 51:139–186, 2004. doi:10.1145/972639.972641.

4 M. Aigner and G. M.Ziegler. Three applications of euler's formula: Pick's theorem. In *Proofs from THE BOOK (6th ed.)*, pages 93–94. Springer, 2018.

5 B. Aronov and M. Sharir. Cutting circles into pseudo-segments and improved bounds for incidences. *Discrete Comput. Geom.*, 28:475–490, 2002.

6 E. Bombieri and J. Pila. The number of integral points on arcs and ovals. *Duke Math. Journal*, 59(2):337–357, 1989.

7 G. Elekes. *Sums versus products in number theory, algebra and Erdős geometry*, volume 11 of *Bolyai Soc. Math. Stud.*, pages 241–290. János Bolyai Math. Soc., Budapest, 2002.

8 B. Janzer, O. Janzer, A. Methuku, and G. Tardos. Tight bounds for intersection-reverse sequences, edge-ordered graphs and applications, 2024. arXiv:2411.07188.

9 T. Kóvari, V. Sós, and P. Turán. On a problem of K. Zarankiewicz. *Colloquium Mathematicae*, 3(1):50–57, 1954. URL: http://eudml.org/doc/210011.

10 A. Marcus and G. Tardos. Intersection reverse sequences and geometric applications. *Journal of Combinatorial Theory, Series A*, 113(4):675–691, 2006. doi:10.1016/j.jcta.2005.07.002.

11 J. Pach and M.Sharir. Geometric incidences. In J. Pach, editor, *Towards a Theory of Geometric Graphs*, volume 342 of *Contemp. Math.*, pages 185–223. Amer. Math. Soc. Press, 2004.

12 M. Sharir and J. Zahl. Cutting algebraic curves into pseudo-segments and applications. *J. Comb. Theory A*, 150:1–35, 2017. doi:10.1016/J.JCTA.2017.02.006.

13 A. Sheffer. *Polynomial Methods and Incidence Theory*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 2022. doi:10.1017/9781108959988.

14 E. Szemerédi and W. T. Trotter Jr. Extremal problems in discrete geometry. *Combinatorica*, 3:381–392, 1983. doi:10.1007/BF02579194.

Compressibility Measures and Succinct Data Structures for Piecewise Linear Approximations

Paolo Ferragina
Department L'EMbeDS, Sant'Anna School of Advanced Studies, Pisa, Italy
Department of Computer Science, University of Pisa, Italy

Filippo Lari[1]
Department of Computer Science, University of Pisa, Italy

Abstract

We study the problem of deriving compressibility measures for *Piecewise Linear Approximations* (PLAs), i.e., error-bounded approximations of a set of two-dimensional *increasing* data points using a sequence of segments. Such approximations are widely used tools in implementing many *learned data structures*, which mix learning models with traditional algorithmic design blocks to exploit regularities in the underlying data distribution, providing novel and effective space-time trade-offs.

We introduce the first lower bounds to the cost of storing PLAs in two settings, namely *compression* and *indexing*. We then compare these compressibility measures to known data structures, and show that they are asymptotically optimal up to a constant factor from the space lower bounds. Finally, we design the first data structures for the aforementioned settings that achieve the space lower bounds plus small additive terms, which turn out to be *succinct* in most practical cases. Our data structures support the efficient retrieval and evaluation of a segment in the (compressed) PLA for a given x-value, which is a core operation in any learned data structure relying on PLAs.

As a result, our paper offers the first theoretical analysis of the maximum compressibility achievable by PLA-based learned data structures, and provides novel storage schemes for PLAs offering strong theoretical guarantees while also suggesting simple and efficient practical implementations.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Data compression; Information systems → Information retrieval

Keywords and phrases Piecewise Linear Approximations, Succinct Data Structures, Lower Bounds

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.31

Related Version *Full Version*: `https://arxiv.org/abs/2509.07827` [8]

Funding The work of P.F. was partially supported by the Alfred P. Sloan Foundation with the grant #G-2025-25193 (`sloan.org`). The work of both authors has been supported by the Next-GenerationEU – National Recovery and Resilience Plan (Piano Nazionale di Ripresa e Resilienza, PNRR) – Project: "SoBigData.it - Strengthening the Italian RI for Social Mining and Big Data Analytics" – Prot. IR0000013 – Avviso n. 3264 del 28/12/2021; and by the European Union – Horizon 2020 Program under the scheme "INFRAIA-01-2018-2019 – Integrating Activities for Advanced Communities", Grant Agreement n. 871042, "SoBigData++: European Integrated Infrastructure for Social Mining and Big Data Analytics" (`http://www.sobigdata.eu`).

1 Introduction

The rapid growth of data has led to the development of novel techniques for designing data structures that leverage regularities in the underlying data distribution to deliver faster or more succinct solutions to proper query operations. This novel line of research, known as *learned data structures* [15], integrates machine learning models with traditional algorithmic techniques to achieve improved space occupancy and query time performance.

[1] Corresponding author

© Paolo Ferragina and Filippo Lari;
licensed under Creative Commons License CC-BY 4.0

A widely used tool in the field of learned data structures is the so-called *Piecewise Linear Approximation* (PLA), an error-bounded approximation of a two-dimensional set of points using a sequence of segments [20]. The common use of PLAs is due to their ability to capture linear trends in the input data, which frequently occur in many practical applications, while using little space and providing fast inference time. In this work, we focus on PLAs arising in two settings: (1) the *compression* setting, where the input sequence of points is increasing in both coordinates but the x-values are also consecutive; and (2) the *indexing* setting, where instead the y-values of the input sequence are consecutive. These two settings are general enough to cover a broad range of applications, including *rank* and *select* dictionaries [7, 2], time series compression [14], string indexing [24], one-dimensional indexing [15, 13], multi-dimensional indexing [17, 4], range minimum queries [9], monotone minimal perfect hashing [10], and range filtering [23], to mention a few.

Prior work concerning the study of theoretical properties of PLAs has primarily focused on providing *upper bounds* on the optimal number of segments ℓ as a function of the sequence length n and the maximum allowed error ε, that is the maximum vertical distance between the y-coordinate of any input point and its *covering* segment. Such a relationship is fundamental from a theoretical perspective since it can explain the surprising performance of learned data structures and also allow a theoretical comparison with their classical counterparts. In this context, it is worth mentioning the pioneering study of Ferragina et al. [11] showing that in the indexing case under the assumption that the gaps between consecutive keys are independently and identically distributed random variables, with constant mean and variance, the optimal number of segments ℓ is $\Theta(n/\varepsilon^2)$ with high probability. Later works [25, 3] provided better upper bounds, but they only considered PLAs formed by *horizontal segments*.

As far as *lower bounds* are concerned, we mention the work of Zeighami and Shahabi [26] which was the first to limit from below the model size (e.g., the number of segments ℓ in a PLA or the number of neurons in a neural network) of any learned data structure achieving a given desired accuracy ε for a wide range of problems arising in database systems such as one- and multi-dimensional indexing, cardinality estimation, and range-sum estimation.

In our work, we assume that the segments have *arbitrary slopes* and the number of segments ℓ forming an optimal PLA is known (e.g., computed by [20]), and thus provide the first information-theoretic lower bound to the storage complexity of that PLA in both the compression and the indexing settings (see Theorems 6 and 8). We then compare two representative data structures against these lower bounds, namely the LA-vector by Boffa et al. [2] (for the compression setting) and the PGM-index by Ferragina & Vinciguerra [13] (for the indexing setting), showing that both are optimal up to a constant factor. Finally, we conclude our work by designing two novel data structures for the aforementioned compression and indexing settings, that provide the same asymptotic query time as the LA-vector and the PGM-index, but with small additive terms to the introduced lower bounds.

2 Background

We assume the transdichotomous word RAM model where the word size w matches the logarithm of the problem size and arithmetic, logic, and bitwise operations, as well as access to w-bit memory cells, take $O(1)$ time. To simplify, we denote with $\log x$ the logarithm in base 2 of x, and we assume that logarithms hide their ceiling and thus return integers.

Given a static sorted sequence $S = 1 \leq x_1 < x_2 < \cdots < x_n \leq u$ of n elements drawn from a finite integer universe U of size u, we consider the following fundamental operations:

- $rank(S, x)$, which for $x \in U$ returns the number of S's elements that are smaller or equal to x, i.e. $|\{x_i \in S \,|\, x_i \leq x\}|$.
- $select(S, i)$, which for $i \in \{1, 2, \ldots, n\}$, returns the element in S of position i, i.e. $rank(S, x) = i$. We assume $select(S, 0) = 0$.

If S is represented with its characteristic bit vector B_S, namely, a bit vector of length u such that $B_S[i] = 1$ if $i \in S$, and $B_S[i] = 0$ otherwise, then supporting such operations is equivalent to support the $rank_1(B_S, x)$, which yields the number of 1s before position x, and $select_1(B_S, i)$, which gives the position of the i-th 1 in B_S. These operations have been studied extensively with many practical and theoretical results, and are at the core of any succinct or compressed data structure (see [18] for a comprehensive treatment). For our purposes, we briefly mention that they can be supported in $O(1)$ time while using only $o(u)$ additional bits on top of the underlying bit vector.

The information-theoretic lower bound to store S is $\mathcal{B}(u, n) = \log \binom{u}{n}$. Such value is related to the zero-order empirical entropy of its characteristic bit vector B_S, defined as $u\mathcal{H}_0(B_S) = n \log \frac{u}{n} + (u - n) \log \frac{u}{u-n}$. In fact, with simple arithmetic manipulations one obtains $\mathcal{B}(u, n) = u\mathcal{H}_0(B_S) - O(\log u)$. The best upper bound to represent S with the minimum redundancy with respect to $\mathcal{B}(u, n)$, and while still being able to perform *rank* and *select* in optimal constant time, is given by the following data structure due to Pătraşcu [21]:

▶ **Theorem 1** (Succincter). *For any $c > 0$, a monotone sequence S of n positive integers drawn from a universe of size u can be stored using $\mathcal{B}(u, n) + O(u/\log^c(u/c)) + O(u^{3/4} \, poly \log u)$ bits and support rank and select in $O(c)$ time.*

Theorem 1 implies constant time *rank* and *select* operations in $\mathcal{B}(u, n) + O(u/\log^c u)$ space, which essentially matches the lower bound of Pătraşcu & Viola [22]. To further reduce the space redundancy on top of $\mathcal{B}(u, n)$, one must allow for slower *rank* or *select* operations. A prominent example is given by the Elias-Fano encoding scheme [6, 5], which attains the following space-time trade-off for the aforementioned operations [18]:

▶ **Theorem 2** (Elias-Fano encoding). *A monotone sequence S of n positive integers drawn from a universe of size u can be stored using $2n + n \log \frac{u}{n} + o(n)$ bits while supporting select in $O(1)$ time and rank in $O(\min\{\log \frac{u}{n}, \log n\})$ time.*

Indeed, the space usage of Theorem 2 can be expressed in terms of $\mathcal{B}(u, n)$ as follows:

$$2n + n \log \frac{u}{n} + o(n) = u\mathcal{H}_0(B_S) + O(n) = \mathcal{B}(u, n) + O(n + \log u) \tag{1}$$

Thus obtaining a lower redundancy term with respect to Theorem 1 whenever S is very sparse, which, however, requires giving up the constant time *rank* operation. The literature offers other trade-offs (e.g., [18, 19]) that could be adopted in our solution, leading to corresponding time-space bounds. However, for the sake of presentation, in the remainder of the paper we focus on the results provided in Theorems 1 and 2.

A new line of research has recently started to investigate novel methods for designing algorithms and data structures that leverage regularities in the underlying data to deliver faster or more succinct solutions to a wide variety of problems. Most of these methods rely on a technique known as *Piecewise Linear Approximation* (PLA), which involves approximating a sequence of two-dimensional points using a set of segments.

▶ **Definition 3.** *Given a set of n two-dimensional data points $P \subseteq \mathbb{R}^2$, a Piecewise Linear Approximation of P with maximum error $\varepsilon \geq 1$ is the smallest set of non-overlapping segments $\{s_1, s_2, \ldots, s_\ell\}$ that entirely covers P, ensuring that every point in P lies within a maximum vertical distance ε from its corresponding segment.*

A PLA can always be computed optimally, hence producing the minimum number of segments ℓ in time proportional to the sequence length using an algorithm by O'Rourke [20].

Here, we focus on PLAs computed from monotone sequences of two-dimensional data points, in which a point (x_i, y_i) is smaller than (x_{i+1}, y_{i+1}) because it has a strictly smaller abscissa $x_i < x_{i+1}$ and a smaller ordinate $y_i \leq y_{i+1}$. In particular, we consider two specific scenarios: (1) the compression setting, where PLAs are computed on sequences having consecutive x-values (i.e., $x_{i+1} = x_i + 1$, with $x_1 = 1$), and (2) the indexing setting, in which PLAs are computed on sequences having consecutive y-values (i.e. $y_{i+1} = y_i + 1$, with $y_1 = 1$). As already mentioned, despite the names, such scenarios are quite general and encompass a wide range of applications. To get a practical sense of the usage of PLAs in the design of learned data structures, we briefly outline two prominent examples.

In the design of learned and compressed *rank* and *select* dictionaries [2], the input sequence $S = 1 \leq x_1 < x_2 < \cdots < x_n \leq u$, is first mapped to a Cartesian plane representation where the x-axis corresponds to the sequence of positions $1, 2, \ldots, n$, while the y-axis represents the sequence values, namely x_is. This transformation converts S into a set of monotonically increasing two-dimensional points: $\{(i, x_i) \,|\, x_i \in S\}$. The key innovation of this approach is that the Cartesian plane representation reveals inherent regularities in the underlying sequence S. A PLA is then computed from such a sequence of points, and compression is achieved by storing for each point (i, x_i) the absolute difference between x_i and the approximation of the segment for position i using just $\log(2\varepsilon + 1)$ bits. Performing a $select(S, i)$ operation then reduces to first finding the segment covering the abscissa value i, computing the approximate sequence value from such segment in i, and finally adding the correction term stored in position i to retrieve the exact value. A $rank(S, x)$ operation can be naively solved by binary searching on the *select* values, even if faster alternatives do exist [2].

In the design of learned one-dimensional indexes [13], given the input sequence $S = 1 \leq x_1 < x_2 < \cdots < x_n \leq u$, one performs the same mapping above onto a Cartesian plane representation. In this case, the x-axis contains the sequence values, while the y-axis corresponds to their respective positions $1, 2, \ldots, n$. Again, this transformation converts S into a set of monotonically increasing two-dimensional data points: $\{(x_i, i) \,|\, x_i \in S\}$. The data structure then consists of a PLA computed from such a set of points and a search for a given key x proceeds in three steps. First, the segment covering x is identified through a binary search on the first keys covered by each segment. Second, the approximate position i of x is obtained from such a segment. Third, since the underlying sequence is monotonically increasing and the approximate position i of x is precise up to an error ε, it is *corrected* via a (binary) search in the interval $[i - \varepsilon, i + \varepsilon]$. Such queries can be sped up by applying the same construction recursively to the sequence of first keys covered by each segment until a single one remains (see e.g. [13, 15]). This approach implements the index as a hierarchical structure, forming a tree of linear models. Consequently, searches involve a top-down traversal of the tree, trading faster queries for an increased space usage.

3 Related Work

In the following, we review existing methods for the lossless compression of PLAs, focusing on the $predict(x)$ operation, which, given a value lying on the x-axis, identifies the segment covering x and computes the corresponding y-value *predicted* by that segment.

Ferragina & Vinciguerra introduced the PGM-index [13], the first solution in the indexing case based on the optimal PLA computed by the O'Rourke algorithm [20], i.e., the one consisting of the minimum number of segments ℓ providing an ε-approximation of the input points. In particular, they designed two compression methods for the segments. The first one hinges on the observation that, in the indexing setting, the intercepts are monotonically

increasing. Denoting with β_i the intercept of the i-th segment and with y_i its first covered y-value, this easily follows since $\beta_i \in [y_i - \varepsilon, y_i + \varepsilon]$ and $\beta_{i+1} \in [y_{i+1} - \varepsilon, y_{i+1} + \varepsilon]$, but $y_i + \varepsilon \leq y_{i+1} - \varepsilon$ as in this particular case a segment covers at least 2ε keys [13, Lemma 2]. Exploiting this, they convert the monotone sequence of floating-point intercepts into integers and store them with the data structure of Okanohara & Sadakane [19]. But, they stored uncompressed the first covered keys and slopes of each segment. For the sake of comparison, we provide the space usage for the PLA composing just the first layer of the PGM index. Since the universe of the underlying sequence is u, and each slope can be expressed as a rational number with a numerator and denominator respectively of $\log n$ and $\log u$ bits, they obtained the following result which we specialize via two approaches, one based on binary search (as proposed in [13]), and one using the data structure of Pătraşcu (see Theorem 1):

▶ **Lemma 4** (PGM-index [13]). *The PLA composing the first layer of the PGM-index can be stored using:*

- $\ell\,(1.92 + \log \frac{n^2}{\ell} + 2\,\log u + o(1))$ *bits, while supporting the predict operation in* $O(\log \ell)$ *time via binary search.*
- $\ell\,(1.92 + \log \frac{n^2}{\ell} + \log u + o(1)) + \log \binom{u}{\ell} + O(u/\log^c u)$ *bits for any constant* $c > 0$*, while supporting the predict operation in* $O(c)$ *time.*

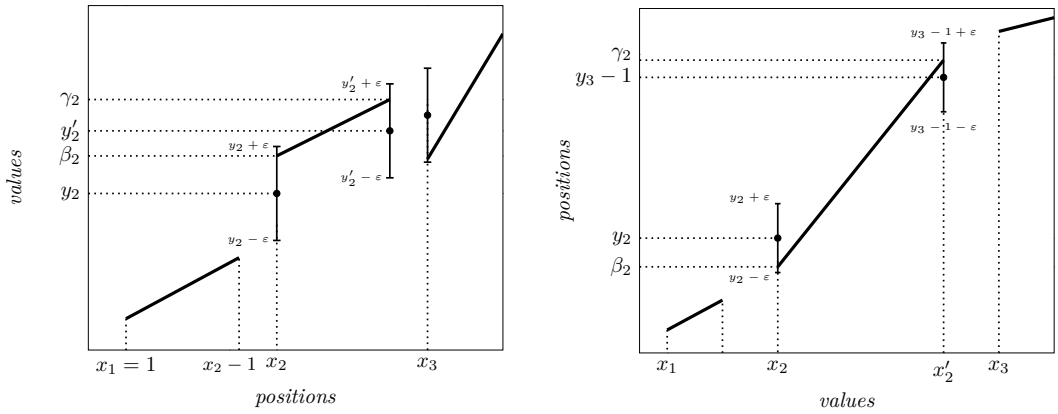
The second compression method targets the segment's slopes, and is based on the observation that the O'Rourke algorithm does not compute a single segment but a whole family of segments whose slopes identify an interval of reals. The compressor then works greedily (and optimally) by trying to assign the same slope to most segments of the PLA, thus reducing the number of slopes to be fully represented. This is very effective on some real-world datasets, however, the authors did not provide an analysis of its space occupancy.

Moving to the compression setting, Boffa et al. introduced the LA-vector [2], the first learned and compressed *rank* and *select* dictionary based on the optimal PLA. Their compression scheme for segments hinges on the assumption that the sequence of intercepts is monotonically increasing, which is not necessarily the case in general. This is because, in the compression setting, segments may cover fewer than 2ε points, as the y-value can increase by more than one between consecutive x-values. Ferragina & Lari [9] recently showed how to waive that assumption with a negligible impact on space occupancy, obtaining the following bound (we drop the cost of cn bits for storing the correction vector C in Theorem 3.2 of [2], and we add the solution based on the data structure of Pătraşcu):

▶ **Lemma 5** (LA vector, Theorem 2.3 in [2] and Theorem 5 in [9]). *The PLA of the LA-vector can be stored using*

- $\ell\,(2\log \frac{u}{\ell} + \log \frac{n}{\ell} + 6 + 2\log(2\varepsilon + 1) + o(1))$ *bits, while supporting the predict operation in* $O(\log \ell)$ *time via binary search.*
- $\ell\,(2\log \frac{u}{\ell} + 4 + 2\log(2\varepsilon + 1) + o(1)) + \log \binom{n}{\ell} + O(n/\log^c n)$ *bits for any constant* $c > 0$*, while supporting the predict operation in* $O(c)$*.*

Lastly, Abrav and Medvedev proposed the PLA-index [1] in a Bioinformatics application where the x- and y-values of the underlying sequence are increasing but not necessarily contiguous. They took an orthogonal approach to PLA compression by modifying the O'Rourke algorithm, enforcing each segment to start from the ending point of the previous one. This allows them to derive a more compact encoding, which may produce a suboptimal number of segments $\ell' \geq \ell$. As their approach does not guarantee optimality and targets different sequence types, we defer comparison to the journal version.



(a) **(b)**

Figure 1 The key parameters of PLAs arising in the two considered settings. Figure 1a shows an example in the compression case. The values on the x-axis are consecutive, the first segment always begins at position 1, and each segment starts at an x-value immediately after the ending of the previous one. Intercepts and the last y-values of each segment stay within a range of size $2\varepsilon + 1$ from the first and last covered values of the sequence. Notice how the intercepts do not necessarily form an increasing sequence. Figure 1b considers the indexing case, which is similar to the compression setting except that the values on the y-axis are consecutive. Hence, the last covered y-value of a segment is implicitly determined by the first covered y-value of the next one, and the intercepts β_is form a monotone sequence as $y_i + \varepsilon \leq y_{i+1} - \varepsilon$ since a segment covers at least 2ε keys [13, Lemma 2].

We conclude by noticing that none of these results focused on studying lower bounds to the cost of storing PLAs (possibly in compressed form). Therefore, the first novel contribution of our work lies in deriving the first compressibility measures for PLAs in the two aforementioned settings. It will turn out that the storage schemes for PLAs of the LA-vector and the PGM-index are asymptotically optimal up to a constant factor. Hence, our second contribution is to design the first data structures for the compression and indexing settings that achieve the space lower bounds plus small additive terms, which are *succinct* in most practical cases.

4 Compressibility Measures for PLAs

We begin by observing that, in general, the segments of a PLA, whether in the compression or indexing setting, have floating-point components. As discussed in Section 3, some existing compression schemes assume that segments either start or end at an integer y-coordinate to exploit integer compression techniques. In our context, we work under the same assumption for two reasons: first, when establishing a lower bound on the space required to represent a PLA, it is important to note that the class of PLAs with floating-point components is larger than that of PLAs with integer components. Consequently, any lower bound proven for the latter also applies to the former. Second, as proved by Ferragina & Lari [9] this reduction has a negligible impact on the approximation error of the resulting PLA, and in fact, it only increases that error by a constant 3. Therefore, without loss of generality, we can safely restrict our analysis to PLAs having segments with integer starting and ending ordinates.

Under this assumption, given the optimal number of segments ℓ, the approximation error ε, the universe size u, and the length of the sequence n, we consider the problem of counting the number of possible PLAs that can be constructed using these parameters, in both the compression and indexing cases. Then, by a simple information-theoretic argument, taking

the logarithm of such quantities gives a lower bound on the number of bits required to represent any PLA in the considered settings, and this provides a compressibility measure that any compression scheme should strive to achieve. Starting from PLAs arising in the compression setting, we prove the following theorem:

▶ **Theorem 6.** *Let $\mathcal{C}(\ell, \varepsilon, u, n, \mathbf{y})$ be the set of PLAs of ℓ segments having maximum error ε, covering a monotone sequence of n elements drawn from a universe of size u in the compression setting, and having $\mathbf{y} = y_1 \leq y_2 \leq \cdots \leq y_\ell$ as the first covered y-values. Then:*

$$|\mathcal{C}(\ell, \varepsilon, u, n, \mathbf{y})| = \binom{n-\ell-1}{\ell-1}\binom{u+\ell-1}{\ell}\left(\prod_{i=1}^{\ell-1}(y_{i+1}-y_i+1)\right)(2\varepsilon+1)^{2\ell}$$

Proof. Let us denote with x_i the first position covered by the i-th segment, and let us start by constraining each x_i in such a way as to obtain a valid PLA with error ε. Since the first segment always starts from the first position, it is $x_1 = 1$. Each segment covers at least two elements because there is always a segment connecting two points without incurring any error (i.e., the one connecting them), hence $x_{i+1} - x_i \geq 2$ (recall that in the compression setting the x-values are strictly increasing and contiguous).

To ease our counting, we introduce the following change of variables $\bar{x}_i = x_i - (i-1)$. In this way, $\bar{x}_2 = x_2 - 1 \geq 2$ and $\bar{x}_\ell = x_\ell - (\ell - 1) \leq n - \ell$, since $x_2 \geq 3$ and $x_\ell \leq n - 1$. Additionally, $\bar{x}_{i+1} - \bar{x}_i \geq 1$ (since the $\bar{x}_i$s are distinct) and thus $x_{i+1} - x_i = \bar{x}_{i+1} - \bar{x}_i + 1 \geq 2$. Therefore, choosing the first position of each segment while simultaneously satisfying the previous constraint on the x_is, is equivalent to choosing $\ell - 1$ distinct values from $\{2, 3, \ldots, n - \ell\}$. The total number of ways to perform this choice is then $\binom{n-\ell-1}{\ell-1}$.

Let β_i and y_i denote, respectively, the intercept of the i-th segment and its first covered y-value (see Figure 1a). Choosing β_i is equivalent to first selecting y_i and then choosing a value within the interval $[y_i - \varepsilon,\, y_i + \varepsilon]$, which contains $2\varepsilon + 1$ elements. In the compression setting, the input sequence may contain repeated values, so consecutive segments can share the same first covered y-value (i.e., $y_{i+1} = y_i$). Thus, choosing the first y-value of each of the ℓ segments amounts to selecting an increasing sequence of ℓ elements from $\{1, 2, \ldots, u\}$. The number of such choices is $\binom{u+\ell-1}{\ell}$. Since each β_i can then be any value of the interval $[y_i - \varepsilon,\, y_i + \varepsilon]$, the total number of possible selections for the ℓ intercepts is $\binom{u+\ell-1}{\ell}(2\varepsilon+1)^\ell$.

Lastly, let γ_i be the y-value given by the i-th segment evaluated on its last covered position (see Figure 1a). We perform the same reasoning as for the intercepts. Let y'_i be the value of the sequence on the last position covered by the i-th segment. Choosing γ_i is equivalent to first selecting y'_i and then choosing any of the values inside $[y'_i - \varepsilon, y'_i + \varepsilon]$. Now, since we are dealing with monotonically increasing sequences, to obtain a valid PLA, we need that $y'_i \geq y_i$ and $y'_i \leq y_{i+1}$ for every $1 \leq i < \ell$, notice that $y'_\ell = u$. Consequently, each y'_i can be independently chosen from the interval $[y_i, y_{i+1}]$, and each γ_i is then selected inside the interval $[y'_i - \varepsilon, y'_i + \varepsilon]$, therefore, the total number of ways to choose the ℓ last y-values of each segment is $\left(\prod_{i=1}^{\ell-1}(y_{i+1} - y_i + 1)\right)(2\varepsilon+1)^\ell$.
Combining these quantities, our claim follows. ◀

Theorem 6 suggests the following lower bound on the number of bits that any lossless compressor for PLAs in the compression setting must output.

▶ **Corollary 7.** *The minimum number of bits to represent any PLA from* $\mathcal{C}(\ell, \varepsilon, u, n, \mathbf{y})$ *is:*

$$\mathcal{B}_{\mathcal{C}}(\ell, \varepsilon, u, n, \mathbf{y}) = \log \binom{n-\ell-1}{\ell-1} + \log \binom{u+\ell-1}{\ell} + \sum_{i=1}^{\ell-1} \log\left(y_{i+1} - y_i + 1\right) + 2\ell \log\left(2\varepsilon + 1\right)$$

Next, we focus on PLAs arising in the indexing setting. Due to space limitations, we present only the main result here and defer the proof to the full version of this paper [8]. We only mention that the proof follows the same strategy as that of Theorem 6, with the only difference that in the indexing setting, each segment (except possibly the last one) covers at least 2ε keys (see [13, Lemma 2]), thus reducing the possible choices for the first covered x- and y-values.

▶ **Theorem 8.** *Let* $\mathcal{I}(\ell, \varepsilon, u, n, \mathbf{x})$ *be the set of PLAs of* ℓ *segments having maximum error* ε*, covering a strictly increasing sequence of* n *elements drawn from a universe of size* u *in the indexing setting, and having* $\mathbf{x} = x_1 < x_2 < \cdots < x_\ell$ *as the first covered* x*-values. Then:*

$$|\mathcal{I}(\ell, \varepsilon, u, n, \mathbf{x})| = \binom{u-\ell(2\varepsilon-1)}{\ell} \binom{n-\ell(2\varepsilon-1)-1}{\ell-1} \left(\prod_{i=1}^{\ell-1} (x_{i+1} - x_i - 1)\right) (2\varepsilon+1)^{2\ell}$$

Theorem 8 gives the following lower bound on the number of bits that any lossless compressor has to produce when applied to PLAs in the indexing setting.

▶ **Corollary 9.** *The minimum number of bits to represent any PLA from* $\mathcal{I}(\ell, \varepsilon, u, n, \mathbf{x})$ *is:*

$$\mathcal{B}_{\mathcal{I}}(\ell, \varepsilon, u, n, \mathbf{x}) = \log \binom{u-\ell(2\varepsilon-1)}{\ell} + \log \binom{n-\ell(2\varepsilon-1)-1}{\ell-1} + \sum_{i=1}^{\ell-1} \log\left(x_{i+1} - x_i - 1\right) + 2\ell \log\left(2\varepsilon + 1\right)$$

As already mentioned, unlike in the compression setting, segments always cover at least 2ε keys (except possibly the last one). Therefore, the lower bound of Corollary 9 can be made independent from the choice of the first covered x-values, by noting that $x_{i+1} - x_i \geq 2\varepsilon$. Nevertheless, this more general form of the lower bound comes at the cost of underestimating the minimum number of bits required to store any PLA in the indexing case when segments span more than 2ε elements.

Having established lower bounds on the space required to represent lossless any PLA in both the compression and indexing settings, we now shift our focus to designing data structures that use space close to $\mathcal{B}_{\mathcal{C}}(\ell, \varepsilon, u, n, \mathbf{y})$ and $\mathcal{B}_{\mathcal{I}}(\ell, \varepsilon, u, n, \mathbf{x})$ while efficiently supporting the core operation of any learned data structure hinging on PLAs (e.g., [13, 2, 23, 7, 14, 9]), namely, returning the y-value *predicted* by the segment covering a given x-value.

5 Succinct Data Structures for PLAs

We begin by establishing constraints on the parameters of a PLA that are commonly observed in many practical applications. Specifically, we assume that $\epsilon = O(1)$, $\ell = o(n)$ and $n = o(u)$. Next, we proceed by formalizing the core operation performed on any PLA in both the compression and indexing settings. In particular, we define the *predict*(x) operation, which, given a value lying on the x-axis, identifies the segment covering x and computes the

corresponding y-value *predicted* by that segment. Designing data structures that efficiently support this operation while achieving space usage close to the lower bounds of Section 4 is of both practical and theoretical interest since it can help to understand what the limits of PLA-based learned data structures are in terms of the redundancy needed to store and query the PLA itself, and possibly improve the performances of such data structures.

At the end of Section 2, we commented on the relation between the current literature and our lower bounds, highlighting that the storage schemes for PLAs of the LA-vector and the PGM-index are *compact* [18]. Due to space constraints, we sketch the main result here; please refer to the full version of this paper for the complete proofs [8]. Starting with the LA-vector (see Section 5.1), it turns out that the storage space of Lemma 5 can be expressed as $\mathcal{B}_{\mathcal{C}}(\ell, \varepsilon, u, n, \mathbf{y})$ with an additive term that is either $O(\ell \log \frac{u}{\ell})$ or $O(\ell \log \frac{u}{\ell} + \frac{n}{\log^c n})$ depending if queries are answered with a binary search or with the data structure of Pătraşcu [21]. In the first case, the overhead is always $O(\mathcal{B}_{\mathcal{C}}(\ell, \varepsilon, u, n, \mathbf{y}))$, while in the second case, that bound holds only when $\ell = \omega(n/(\log u \log^c n))$. As a result, the storage space of the LA-vector is asymptotically optimal up to a constant factor, and thus is compact. Moving to the PGM-index (see the full version of this paper [8]), the additive term on top of the lower bound $\mathcal{B}_{\mathcal{I}}(\ell, \varepsilon, u, n, \mathbf{x})$ is either $\Theta(\ell \log u)$ or $O(\ell \log u + \frac{u}{\log^c u})$, depending on how queries are answered. In the first case, the overhead is always $\Theta(\mathcal{B}_{\mathcal{I}}(\ell, \varepsilon, u, n, \mathbf{x}))$, while in the second case that bound holds only when $\ell = \Omega(u/\log^{c+1} u)$. Therefore, the storage space of the PGM-index is also compact.

Motivated by this, the following sections will describe the design of the first data structures for the compression and indexing settings, achieving small additive terms to the lower bounds of Section 4, which turn out to be succinct in most practical cases, and most importantly provide the same asymptotic query time as the LA-vector and the PGM-index.

5.1 Using space close to $\mathcal{B}_{\mathcal{C}}(\ell, \varepsilon, u, n, \mathbf{y})$

Given a PLA $\{s_1, s_2, \ldots, s_\ell\} \in \mathcal{C}(\ell, \varepsilon, u, n, \mathbf{y})$, we represent a generic segment s_i as a tuple $s_i = (x_i, \beta_i, \gamma_i, y_i, y'_i)$ where x_i is the first covered x-value of the input sequence, β_i is the intercept, γ_i is the last y-value given by the segment, y_i and y'_i are respectively the first and last y-values of the input sequence covered by the i-th segment (see Figure 1a). To encode each segment, we begin by storing a bit sequence X containing the first x-value covered by each segment. Specifically, X encodes in unary the deltas between the first x-values of consecutive segments (i.e., between x_{i+1} and x_i), shifted by a constant to further reduce space usage. Such a sequence is defined as follows:

- $X = 0^{x_2 - x_1 - 2} 1 0^{x_3 - x_2 - 2} 1 \ldots 0^{x_\ell - x_{\ell-1} - 2} 1$. The sequence is well-defined since $x_{i+1} - x_i \geq 2$. The length of X is $\sum_{i=1}^{\ell-1}(x_{i+1} - x_i - 1) = x_\ell - x_1 - (\ell - 1)$, which is upper bounded by $n - \ell - 1$ since $x_1 = 1$ and $x_\ell \leq n - 1$, and contains $\ell - 1$ ones. The i-th element of X can be accessed by noticing that $x_1 = 1$ and for any $i > 1$, $x_i = \mathit{select}_1(X, i - 1) + i$.

Next, we store an integer sequence Y encoding the first covered y-values of each segment. Given our assumption that $\ell = o(n)$, both the X and Y sequences are sparse. Since they are also monotonically increasing by construction, we represent them using the Elias–Fano encoding scheme[2]. By Theorem 2, the space in bits required to store X and Y is:

$$(\ell - 1) \log \left(\frac{n - \ell - 1}{\ell - 1} \right) + \ell \log \left(\frac{u}{\ell} \right) + 4\ell + o(\ell) \tag{2}$$

[2] We turn X into an integer sequence by interpreting each unary code as an integer.

Noting that $\log\binom{u+\ell-1}{\ell} = \log\binom{u}{\ell} + O(\ell)$ (since $\binom{u+\ell-1}{\ell} = \binom{u}{\ell} \cdot \left(\prod_{i=1}^{\ell-1}(u+i)/\prod_{i=1}^{\ell-1}(u-i)\right)$ and the second factor can be bounded by $c^{\ell-1}$ for some constant $c > 1$) and applying the Elias-Fano space bound from Equation 1, we can rewrite Equation 2 in the following form:

$$\log\binom{n-\ell-1}{\ell-1} + \log\binom{u+\ell-1}{\ell} + O(\ell + \log u) \tag{3}$$

Next we consider the sequence of last covered y-values of each segment, i.e. $y'_1, y'_2, \ldots, y'_{\ell-1}$, omitting y'_ℓ as it is always equal to u. Since each $y'_i \in [y_i, y_{i+1}]$, we store the relative values using just $\log(y_{i+1} - y_i + 1)$ bits and concatenate their representation in a single bit vector B, of size $|B| = \sum_{i=1}^{\ell-1} \log(y_{i+1} - y_i + 1)$ bits. To access each y'_i, we maintain an auxiliary array P, where each p_i is the starting index of the binary representation of y'_i inside B. Notice that this technique resembles the *dense pointers* method of Ferragina & Venturini [12]. Array P is both sparse and monotonically increasing, hence we store it with the Elias-Fano encoding scheme using the following number of bits (recall Theorem 2):

$$2(\ell-1) + (\ell-1)\log\left(\frac{\sum_{i=1}^{\ell-1}\log(y_{i+1} - y_i + 1)}{\ell-1}\right) + o(\ell) \tag{4}$$

Using Jensen's inequality, and noticing that $\sum_{i=1}^{\ell-1}(y_{i+1} - y_i + 1) = y_\ell + \ell - 1$ which is at most $u + \ell - 1$ since $y_\ell \le u$, we obtain that $|P| = O(\ell \log\log \frac{u}{\ell})$ bits. Therefore, the sequence of the last covered y-values is encoded as the bit vector B and the Elias-Fano encoding of the array P, using the following number of bits:

$$\left(\sum_{i=1}^{\ell-1}\log(y_{i+1} - y_i + 1)\right) + O(\ell \log\log \frac{u}{\ell}) \tag{5}$$

Lastly, since each segment is an ε-approximation for the points it covers, it holds that $|\beta_i - y_i| \le 2\varepsilon$ and $|\gamma_i - y'_i| \le 2\varepsilon$. Therefore we store each intercept β_i and each last y-value γ_i as deltas with respect to y_i and y'_i inside two arrays Δ_β and Δ_γ using $2\ell\log(2\varepsilon+1)$ bits. Combining this with Equations 3 and 5, and recalling Corollary 7, the space usage of our storage scheme is $\mathcal{B}_{\mathcal{C}}(\ell, \varepsilon, u, n, \mathbf{y}) + O(\ell \log\log \frac{u}{\ell} + \log u)$ bits. That is succinct space, since $O(\ell \log\log \frac{u}{\ell} + \log u) \subseteq o(\mathcal{B}_{\mathcal{C}}(\ell, \varepsilon, u, n, \mathbf{y}))$ because $\mathcal{B}_{\mathcal{C}}(\ell, \varepsilon, u, n, \mathbf{y})$ includes a term that is $\Theta(\ell \log \frac{u}{\ell})$, and we consider the case that $\ell = \Omega(\log u / \log\log u)$.

Without introducing any extra space on top of this representation, performing a *predict*(x) query proceeds as detailed in Algorithm 1. For simplicity, we assume that the segment covering x is never the first one nor the last one. Corner cases can be handled in constant time without altering the query time.

Because of Theorem 2, all the *select* operations on the Elias-Fano encoded sequences X, Y, and P take $O(1)$ time. Reading the binary representation of numbers from B, as well as accessing the deltas inside Δ_β and Δ_γ, requires a $O(1)$ number of bitwise and bit-shifting operations (see [18, Chapter 3] for the technical details), hence $O(1)$ time in the transdichotomous word RAM model we are assuming. The most expensive part of Algorithm 1 is then the predecessor search on X, which dominates the cost of a *predict* query, making the overall time $O(\log \ell)$.

To lower the cost of the predecessor search, and thus achieve faster query times, one must allow for a larger additive term to the lower bound $\mathcal{B}_{\mathcal{C}}(\ell, \varepsilon, u, n, \mathbf{y})$. To address this, we directly store the sequence of first covered x-values as $X = x_2, x_3, \ldots, x_\ell$, whose universe is bounded by $n - 1$ and contains $\ell - 1$ values (recall $x_1 = 1$ in the compression setting).

Algorithm 1 Performs a *predict* operation on our compressed storage scheme for PLAs.

Input : The arrays of first covered x-values and y-values X and Y, the bit-vector B, the array of offsets P, the array of deltas Δ_β and Δ_γ, and a value x.
Output : A value y such that $predict(x) = y$.

$i = pred(X, x)$; `// predecessor search on` X `using a binary search`
$x_i = select_1(X, i-1) + i$;
$x_{i+1} = select_1(X, i) + (i+1)$;
$y_i = select_1(Y, i)$;
$s_i = select_1(P, i)$;
$e_i = select_1(P, i+1) - 1$;
$y'_i = read(B, s_i, e_i)$; `// read the bit sequence` $B[s_i, e_i]$ `representing` y'_i
$\beta_i = y_i + \Delta_\beta[i]$;
$\gamma_i = y'_i + \Delta_\gamma[i]$;
return $\left\lfloor (x - x_i)\frac{(\gamma_i - \beta_i)}{(x_{i+1} - x_i)} \right\rfloor + \beta_i$;

This sequence allows the predecessor of any given position (and, consequently, the index of the segment covering that position) to be computed via a simple *rank* operation. X is then stored using Theorem 1, which guarantees that for any constant $c > 0$, both *rank* and *select* operations take $O(c)$ time while requiring $\log \binom{n-1}{\ell-1} + O\left(\frac{n}{\log^c n}\right)$ bits of space.

Since we assumed that $\ell = o(n)$, we refactor the term $\log \binom{n-1}{\ell-1}$ as follows:

$$\begin{aligned}
\log \binom{n-1}{\ell-1} &= \log \binom{n-\ell-1}{\ell-1} + \log \left(\frac{\prod_{i=1}^{\ell}(n-i)}{\prod_{i=\ell}^{2\ell-1}(n-i)} \right) \\
&\leq \log \binom{n-\ell-1}{\ell-1} + \ell \log \left(\frac{n-1}{n-2\ell+1} \right) \\
&= \log \binom{n-\ell-1}{\ell-1} + O(\ell)
\end{aligned}$$

Where the inequality is obtained by upper-bounding each term in the numerator by $n-1$ and lower-bounding each term in the denominator by $n - 2\ell + 1$. The space usage is then:

$$\mathcal{B}_{\mathcal{C}}(\ell, \varepsilon, u, n, \mathbf{y}) + O(\ell \log\log \frac{u}{\ell} + \frac{n}{\log^c n}) \tag{6}$$

We now show that the space usage of this second data structure is succinct. To this end, we use the well-known upper and lower bounds on the logarithm of the binomial (i.e., $k \log \frac{m}{k} \leq \log \binom{m}{k} \leq k \log (\frac{em}{k})$, where $e \approx 2.718$ is the Euler's number, and thus $\log \binom{m}{k} = \Theta(k \log \frac{m}{k})$), to rewrite the lower bound of Corollary 7 as follows:

$$\mathcal{B}_{\mathcal{C}}(\ell, \varepsilon, u, n, \mathbf{y}) = \Theta(\ell(\log \frac{u}{\ell} + \log \frac{n}{\ell})) = \Theta(\ell \log \frac{u}{\ell}) \tag{7}$$

Where the Ω-term comes from the first two additive terms in the formula of $\mathcal{B}_{\mathcal{C}}(\ell, \varepsilon, u, n, \mathbf{y})$ and the approximation of the logarithm of the binomial coefficient written above. The $O()$-term comes similarly and from upper bounding the summation in Corollary 7 using Jensen's inequality, as already done in Section 5.1.

Focusing on the additive term of Equation 6, it is easy to see that $O(\ell \log\log \frac{u}{\ell} + \frac{n}{\log^c n}) \subseteq o(\ell \log \frac{u}{\ell} + \frac{n}{\log^c n})$, by considering $\ell = \omega(n/(\log u \log^c n))$ which implies that $\ell \log \frac{u}{\ell} = \omega(\frac{n}{\log^c n})$. Therefore, from Equation 7, it is $O(\ell \log\log \frac{u}{\ell} + \frac{n}{\log^c n}) \subseteq o(\mathcal{B}_{\mathcal{C}}(\ell, \varepsilon, u, n, \mathbf{y}))$ for those values of ℓ that are $\omega(n/(\log u \log^c n))$ and do not violate our initial assumption that $\ell = o(n)$.

As a result, we proved the following theorem, which gives data structures able to store any PLA in the compression setting using space close to the lower bound of Corollary 7 while supporting efficient *predict* queries.

▶ **Theorem 10.** *There exist data structures storing any PLA from* $\mathcal{C}(\ell, \varepsilon, u, n, \mathbf{y})$ *as follows:*

- *Using* $\mathcal{B}_{\mathcal{C}}(\ell, \varepsilon, u, n, \mathbf{y}) + O(\ell \log\log \frac{u}{\ell} + \log u)$ *bits of space, while supporting predict in* $O(\log \ell)$ *time.*
- *Using* $\mathcal{B}_{\mathcal{C}}(\ell, \varepsilon, u, n, \mathbf{y}) + O(\ell \log\log \frac{u}{\ell} + n/\log^c n)$ *bits of space for any constant* $c > 0$*, while supporting predict in* $O(c)$ *time.*

We conclude by comparing Theorem 10 with the storage scheme for PLAs of the LA-vector (see Lemma 5). We start by considering the one solving the *predict* queries with a binary search on the first covered positions, reporting its space usage in bits here to facilitate comparison:

$$\ell(2\log\frac{u}{\ell} + \log\frac{n}{\ell} + 6 + 2\log(2\varepsilon+1) + o(1)) \tag{8}$$

We notice that the term $2\ell\log(2\varepsilon+1)$ is shared by both Corollary 7 and Equation 8, hence we only focus on the remaining parts. Since we assumed that $\ell = o(n)$, the terms $\ell\log\frac{u}{\ell}$ and $\ell\log\frac{n}{\ell}$ appearing in Equation 8 can be respectively bounded as $\log\binom{u+\ell-1}{\ell} + O(\ell + \log u)$ and $\log\binom{n-\ell-1}{\ell-1} + O(\ell + \log n)$ (see Section 2 and 5.1). Consequently, Equation 8 can be rewritten in the following form, which is closer to the one of Corollary 7.

$$\log\binom{n-\ell-1}{\ell-1} + \log\binom{u+\ell-1}{\ell} + \ell\log\frac{u}{\ell} + 2\ell\log(2\varepsilon+1) + O(\ell + \log u) \tag{9}$$

Lastly, the summation $\sum_{i=1}^{\ell-1}\log(y_{i+1} - y_i + 1)$ appears in the lower bound of Corollary 7, but not in Equation 9 which in turns contain also the additive terms $\ell\log\frac{u}{\ell} + O(\ell + \log u)$.

Now, the minimum of the summation is $\ell - 1$, and this occurs when each segment covers exactly two positions. We rule out the case in which it is zero, as this would force $y_{i+1} = y_i$, thus only considering PLAs formed by a single segment. In this way, the gap between the lower bound and the storage space of the LA-vector is thus $\Theta(\ell\log\frac{u}{\ell}) = \Theta(\mathcal{B}_{\mathcal{C}}(\ell, \varepsilon, u, n, \mathbf{y}))$. Hence, the storage scheme for PLAs of the LA-vector with the correction of Ferragina & Lari is *compact*, because it is up to a constant factor from the optimal. Therefore, our first data structure in Theorem 10 offers an even smaller additive term to the optimal space usage, while having the same asymptotic query time. Specifically, assuming $\ell = \Omega(\log u/\log\log u)$, it exponentially reduces the overhead, from $O(\ell\log\frac{u}{\ell})$ to $O(\ell\log\log\frac{u}{\ell})$.

Similarly, an analogous result holds for the two data structures described in Lemma 5 and Theorem 10, which answer queries in $O(c)$ time for any constant $c > 0$. In this case, it is sufficient to notice that term due to the data structure of Pătraşcu is shared by both, and the additive term of the LA-vector becomes $O(\ell\log\frac{u}{\ell} + \frac{n}{\log^c n})$. Therefore, using the same argument we used for proving the succinctness of the second data structure in Theorem 10, it is possible to show that for $\ell = \omega(n/\log u\log^c n)$ the storage scheme of the LA-vector is again compact, while ours is succinct, and thus offering a lower overhead to the optimal space usage without altering its asymptotic query time.

In conclusion, we remark that, beyond its theoretical appeal, the first data structure of Theorem 10 is also practical, as it leverages techniques with highly engineered implementations, such as the Elias-Fano encoding of monotone sequences [16] and arrays of fixed- and variable-size elements [18]. Indeed, under the same assumption that $\ell = \Omega(\log u/\log\log u)$, our storage scheme asymptotically requires just $O(\log\log\frac{u}{\ell})$ bits per segment over the theoretical

minimum, while still providing practically fast *predict* queries. Conversely, the second data structure is mostly theoretical, and it aims at showing how close we can approach the lower bound of Corollary 7 while still supporting (optimal) constant time *predict* queries.

5.2 Using space close to $\mathcal{B}_{\mathcal{I}}(\ell, \varepsilon, u, n, \mathbf{x})$

The compression and indexing settings are closely related, the only difference lies in which axis is constrained to have consecutive values. Indeed, this similarity can also be observed in the two lower bounds of Corollary 7 and 9 sharing the same form. As a result, the design of a data structure that achieves space usage close to the lower bound for the indexing setting, while supporting efficient *predict* queries, builds upon the same ideas and algorithmic techniques introduced in Section 5.1. Due to space constraints, we present only the main result in this section, deferring the complete to the full version of this paper [8].

▶ **Theorem 11.** *There exist data structures storing any PLA from $\mathcal{I}(\ell, \varepsilon, u, n, \mathbf{x})$ as follows:*

- *Using $\mathcal{B}_{\mathcal{I}}(\ell, \varepsilon, u, n, \mathbf{x}) + O(\ell \log\log \frac{u}{\ell} + \log u)$ bits of space, while supporting the predict operation in $O(\log \ell)$ time.*
- *Using $\mathcal{B}_{\mathcal{I}}(\ell, \varepsilon, u, n, \mathbf{x}) + O(\ell \log\log \frac{u}{\ell} + u/\log^c u)$ bits of space for any constant $c > 0$, while supporting the predict operation in $O(c)$ time.*

The storage scheme of the PGM-index is optimal up to a constant factor. Considering the first data structure of Lemma 4, it can be expressed as $\mathcal{B}_{\mathcal{I}}(\ell, \varepsilon, u, n, \mathbf{x}) + \Theta(\ell \log u)$. As a result, Theorem 11 introduces the first data structure storing any PLA in the indexing setting within lower-order terms to $\mathcal{B}_{\mathcal{I}}(\ell, \varepsilon, u, n, \mathbf{x})$, under the assumption that $\ell = \Omega(\log u / \log\log u)$, with the same asymptotic query time (see the full version of this paper [8]).

The same observations of Section 5.1 regarding the practicality of our data structures hold. The first one is the most practical, while the second is just theoretical and shows how close the lower bound can be approached while answering queries in (optimal) constant time.

6 Conclusions

In this work, we addressed the problem of deriving the first compressibility measures for PLAs of monotone sequences in two distinct settings: *compression* and *indexing*, which are general enough to cover a broad range of applications. The measures we obtained provide a theoretical foundation for understanding the space requirements of any learned data structure relying on PLAs in any of its components.

Based on our compressibility measures, we analyzed two representative data structures for storing PLAs, namely the one of the LA-vector (compression setting) and the one of the PGM-index (indexing setting), showing that they are both optimal up to a constant factor.

Motivated by this study, we proposed two novel data structures for the above settings, which turn out to be succinct (thus optimal up to lower-order terms) in most practical cases while retaining the same asymptotic query time of the LA-vector and PGM-index. Overall, they offer strong theoretical guarantees and suggest efficient practical implementations, providing valuable tools for practitioners and researchers in learned data structures.

Future works could investigate an extension of our analysis to *Piecewise Non-Linear Approximations*, for which successful applications already exist [14]. Another interesting direction, complementary to our work, is the implementation of the proposed data structures with a thorough experimental evaluation of their space-time trade-offs.

References

1 Md. Hasin Abrar and Paul Medvedev. Pla-index: A k-mer index exploiting rank curve linearity. In *24th International Workshop on Algorithms in Bioinformatics, WABI 2024, September 2-4, 2024, Royal Holloway, London, United Kingdom*, volume 312 of *LIPIcs*, pages 13:1–13:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.WABI.2024.13.

2 Antonio Boffa, Paolo Ferragina, and Giorgio Vinciguerra. A learned approach to design compressed rank/select data structures. *ACM Trans. Algorithms*, 18(3):24:1–24:28, 2022. doi:10.1145/3524060.

3 Luis Alberto Croquevielle, Guang Yang, Liang Liang, Ali Hadian, and Thomas Heinis. Beyond logarithmic bounds: Querying in constant expected time with learned indexes. In *28th International Conference on Database Theory, ICDT 2025, March 25-28, 2025, Barcelona, Spain*, volume 328 of *LIPIcs*, pages 19:1–19:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPICS.ICDT.2025.19.

4 Jialin Ding, Vikram Nathan, Mohammad Alizadeh, and Tim Kraska. Tsunami: A learned multi-dimensional index for correlated data and skewed workloads. *Proc. VLDB Endow.*, 14(2):74–86, 2020. doi:10.14778/3425879.3425880.

5 Peter Elias. Efficient storage and retrieval by content and address of static files. *J. ACM*, 21(2):246–260, 1974. doi:10.1145/321812.321820.

6 Robert M. Fano. On the number of bits required to implement an associative memory. Technical report, Massachusetts Institute of Technology, Project MAC, Computer Structures Group, 1971.

7 Paolo Ferragina, Marco Frasca, Giosuè Cataldo Marinò, and Giorgio Vinciguerra. On nonlinear learned string indexing. *IEEE Access*, 11:74021–74034, 2023. doi:10.1109/ACCESS.2023.3295434.

8 Paolo Ferragina and Filippo Lari. Compressibility Measures and Succinct Data Structures for Piecewise Linear Approximations, 2025. arXiv:2509.07827.

9 Paolo Ferragina and Filippo Lari. FL-RMQ: A learned approach to range minimum queries. In *36th Annual Symposium on Combinatorial Pattern Matching, CPM 2025, June 17-19, 2025, Milan, Italy*, volume 331 of *LIPIcs*, pages 7:1–7:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPICS.CPM.2025.7.

10 Paolo Ferragina, Hans-Peter Lehmann, Peter Sanders, and Giorgio Vinciguerra. Learned monotone minimal perfect hashing. In *31st Annual European Symposium on Algorithms, ESA 2023, September 4-6, 2023, Amsterdam, The Netherlands*, volume 274 of *LIPIcs*, pages 46:1–46:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ESA.2023.46.

11 Paolo Ferragina, Fabrizio Lillo, and Giorgio Vinciguerra. On the performance of learned data structures. *Theor. Comput. Sci.*, 871:107–120, 2021. doi:10.1016/J.TCS.2021.04.015.

12 Paolo Ferragina and Rossano Venturini. A simple storage scheme for strings achieving entropy bounds. *Theor. Comput. Sci.*, 372(1):115–121, 2007. doi:10.1016/J.TCS.2006.12.012.

13 Paolo Ferragina and Giorgio Vinciguerra. The PGM-index: a fully-dynamic compressed learned index with provable worst-case bounds. *Proc. VLDB Endow.*, 13(8):1162–1175, 2020. doi:10.14778/3389133.3389135.

14 Andrea Guerra, Giorgio Vinciguerra, Antonio Boffa, and Paolo Ferragina. Learned compression of nonlinear time series with random access. In *Proc. 41st IEEE International Conference on Data Engineering (ICDE)*, 2025.

15 Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, pages 489–504. ACM, 2018. doi:10.1145/3183713.3196909.

16 Danyang Ma, Simon J. Puglisi, Rajeev Raman, and Bella Zhukova. On elias-fano for rank queries in fm-indexes. In *31st Data Compression Conference, DCC 2021, Snowbird, UT, USA, March 23-26, 2021*, pages 223–232. IEEE, 2021. doi:10.1109/DCC50243.2021.00030.

17 Vikram Nathan, Jialin Ding, Mohammad Alizadeh, and Tim Kraska. Learning multi-dimensional indexes. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, pages 985–1000. ACM, 2020. doi:10.1145/3318464.3380579.

18 Gonzalo Navarro. *Compact Data Structures – A Practical Approach*. Cambridge University Press, 2016.

19 Daisuke Okanohara and Kunihiko Sadakane. Practical entropy-compressed rank/select dictionary. In *Proceedings of the Nine Workshop on Algorithm Engineering and Experiments, ALENEX 2007, New Orleans, Louisiana, USA, January 6, 2007*. SIAM, 2007. doi:10.1137/1.9781611972870.6.

20 Joseph O'Rourke. An on-line algorithm for fitting straight lines between data ranges. *Commun. ACM*, 24(9):574–578, 1981. doi:10.1145/358746.358758.

21 Mihai Pătraşcu. Succincter. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 305–313. IEEE Computer Society, 2008. doi:10.1109/FOCS.2008.83.

22 Mihai Pătraşcu and Emanuele Viola. Cell-probe lower bounds for succinct partial sums. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 117–122. SIAM, 2010. doi:10.1137/1.9781611973075.11.

23 Kapil Vaidya, Tim Kraska, Subarna Chatterjee, Eric R. Knorr, Michael Mitzenmacher, and Stratos Idreos. SNARF: A learning-enhanced range filter. *Proc. VLDB Endow.*, 15(8):1632–1644, 2022. doi:10.14778/3529337.3529347.

24 Youyun Wang, Chuzhe Tang, Zhaoguo Wang, and Haibo Chen. Sindex: a scalable learned index for string keys. In Taesoo Kim and Patrick P. C. Lee, editors, *APSys '20: 11th ACM SIGOPS Asia-Pacific Workshop on Systems, Tsukuba, Japan, August 24-25, 2020*, pages 17–24. ACM, 2020. doi:10.1145/3409963.3410496.

25 Sepanta Zeighami and Cyrus Shahabi. On distribution dependent sub-logarithmic query time of learned indexing. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 40669–40680, 2023. URL: https://proceedings.mlr.press/v202/zeighami23a.html.

26 Sepanta Zeighami and Cyrus Shahabi. Towards establishing guaranteed error for learned database operations. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL: https://openreview.net/forum?id=6tqgL8VluV.

Time-Optimal k-Server

Fabian Frei
CSAIL, MIT, Cambridge, MA, USA

Dennis Komm
Department of Computer Science, ETH Zürich, Switzerland

Moritz Stocker
Department of Computer Science, ETH Zürich, Switzerland

Philip Whittington
Department of Computer Science, ETH Zürich, Switzerland

Abstract

The time-optimal k-server problem minimizes the time spent instead of the distance traveled when serving n requests, appearing one after the other, with k servers in a metric space. The classical distance model was motivated by a hard disk with k heads. Instead of minimal head movements, the time model aims for optimal reading speeds.

This paper provides a lower bound of $2k - 1$ on the competitive ratio of any deterministic online algorithm for the time-optimal k-server problem on a specifically designed metric space. This lower bound coincides with the best known upper bound on the competitive ratio for the classical k-server problem, achieved by the famous work function algorithm. We provide further lower bounds of $k + 1$ for all Euclidean spaces and k for uniform metric spaces.

Our most technical result, proven by applying Yao's principle to a suitable instance distribution, is a lower bound of $k + H_k - 1$ that holds even for randomized algorithms, which contrasts with the best known lower bound for the classical problem, which is polylogarithmic in k.

We hope to initiate further intensive study of this natural problem.

2012 ACM Subject Classification Theory of computation → K-server algorithms

Keywords and phrases k-server problem, optimizing time instead of distance, deterministic and randomized algorithms, Yao's principle

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.32

Related Version *Full Version*: https://doi.org/10.48550/arXiv.2503.05589 [15]

Funding *Fabian Frei*: This research was funded in part by grant number 218001 awarded to Fabian Frei by the Swiss National Science Foundation. Work done in part while at CISPA Helmholtz Center for Information Security and ETH Zurich.

Acknowledgements We thank Peter Rossmanith for his suggestion to study the k-server problem in the time model and Richard Královič for pointing out how the algorithm Robin could be extended to graphs of bounded aspect ratio.

1 Introduction

The *k-server problem*, introduced by Manasse et al. in 1988 [23], has been repeatedly referred to as "the holy grail" of online computation [2, 8, 12]. In particular the k-server conjectures about the best competitive ratios achievable by deterministic and randomized algorithms have inspired intensive research for many decades.

The k-server problem can be considered for any natural k and any given nonempty metric space $\mathcal{M} = (M, d)$ together with an *initial configuration* $C_0 \in M^k$. An instance is a sequence $r_1, \ldots, r_n \in M$ of point requests, revealed one by one. An online algorithm answers each request r_j, knowing only the already revealed requests, with a configuration $C_j \in M^k$

© Fabian Frei, Dennis Komm, Moritz Stocker, and Philip Whittington;
licensed under Creative Commons License CC-BY 4.0

such that $C_j(i) = r_j$ for some i, which we describe as *server* s_i *serving request* r_j. Hence, any solution is a sequence of configurations $C_1, \ldots, C_n \in M^k$ describing the movements of k servers such that each requested point is served by moving a server there. The next request is revealed only once the current request has been served. The *cost* of a solution is traditionally defined as the total of all distances traveled by all servers, yielding the classical, well-researched k-server problem.

Another – no less natural – possibility is to count towards the cost not all but only the *maximum* distance traveled by any server between any two configurations. In this paper, we explore this variant and refer to the different definitions of cost as the *distance model* and the *time model*:

The distance model. This is the well-known classical variant. The cost incurred by the algorithm to serve request r_j is the *sum* of all distances traveled by the servers when changing from configuration C_{j-1} to configuration C_j. The cost of a solution $C_1, \ldots, C_n$ is thus $\sum_{j=1}^{n} \sum_{i=1}^{k} d(C_{j-1}(i), C_j(i))$, the total of all distances moved by all servers. We can imagine paying a fuel cost for all server movements.[1]

The time model. The cost for serving a request is the *maximum* distance traveled by the servers; i.e., the total cost is $\sum_{j=1}^{n} \max_{i=1}^{k} d(C_{j-1}(i), C_j(i))$. Hence, an optimal solution minimizes the total waiting time incurred by the requests until they are served. We thus refer to the problem in this model as the *time-optimal k-server problem*.

In many situations where requests need to be served, it is more important to react as fast as possible rather than moving less. We might consider ambulances or police cars being called to the scene of an emergency, but a good example was already given by the famous seminal paper [23] that used it to introduce the classical k-server problem instead: planning the motions of a hard disk with k heads. It could be argued that rather than caring about minimizing hidden head movements, the typical user wants optimized reading and writing speeds. Despite this, past research has focused almost exclusively on the distance model; see Subsection 1.1.

We hope to initiate a new line of research that focuses on the time model. As usual, the main goal is to determine the best competitive ratios achievable by deterministic and randomized algorithms. We prove several lower bounds in this new model, and design a deterministic algorithm matching these bounds on uniform metric spaces.

1.1 Related Work

In this section, we review known results and open questions for the k-server problem in both models. For the classical distance model, we restrict ourselves to the most important milestones; and at the same time introduce some basic notions and concepts that we make use of later. For the time model, however, we can give a full account.

The Distance Model

We note that on metric spaces with at most k points the algorithm keeping one server on each point is trivially 1-competitive. On a clique with $k + 1$ or more vertices, there is a very simple lower bound of k: for any algorithm, there is an instance that always asks for

[1] We remark that moving more than one server per time step cannot save cost in this model as the algorithm incurs the same cost for any given movement, regardless of when it occurs. However, it is easy in the distance model to convert an algorithm that moves some servers simultaneously into a *lazy* algorithm, which does not do this, but incurs at most the same cost [17].

an unoccupied vertex, causing a cost of 1 with each request, whereas the offline strategy of serving an unoccupied vertex with a server currently at a point that is not among the next $k-1$ requests spends at most a cost of 1 for every k requests. (The k-server problem on cliques of size N is also equivalent to the paging problem with N pages and a cache of size k [6,17].) The lower bound of k can be generalized to all nontrivial metric spaces, that is, to *any* metric space with at least $k+1$ points [6, Thm. 10.1], [17, Thm. 4.4].

Manasse et al. already conjectured that there is a matching upper bound when they introduced the problem in 1988 [23, Conjecture 4].

▶ **Conjecture 1** (*k*-Server Conjecture)**.** *For any metric space and any $k \geq 1$, there is a k-competitive algorithm for the k-server problem in the distance model.*

They also proved the conjecture in the two special cases of only $k = 2$ servers and of k servers on metric spaces with only $k+1$ points [23, Thms. 5, 6]. Note that the journal version of this seminal paper appeared two years later under a different title [24]. This is also how long it took for anyone to establish any upper bound on the competitive ratio that did not grow with the input length n but only with k, albeit with an exponential dependence [13, 14, Thm. 2]. Another four years later, Koutsoupias and Papadimitriou provided the first subexponential bounds by analyzing the so-called *work function algorithm* (WFA, independently proposed as a candidate by multiple researchers [11, Sect. 2.3]), which lowered the upper bound to $2k-1$ [20, Thm. 4.3].

To date, WFA remains the algorithm with the best known competitive ratio for general metric spaces. We are thus left with a constant factor of essentially 2 between the upper and lower bound, despite the decades-long efforts by the research community trying to improve the upper bound or disprove the k-server conjecture. It is conjectured that WFA even attains the best possible competitive ratio of k [20].

But at least for some special metric spaces the k-server conjecture could be proven. A prominent example is the real line. Here, Chrobak et al. [9] were able to give the matching upper bound of k on the competitive ratio by introducing and analyzing the so-called *double coverage* algorithm (DC) with the following, somewhat unintuitive behavior. When a point x is requested, DC moves not only one, but two servers towards x: a closest server to the right of x and a closest server to the left of x. They move at the same speed and both stop once one of them has reached x. This strategy was generalized to trees by Chrobak and Larmore [10]. Lastly, WFA is also known to be k-competitive on metric spaces with at most $k+2$ points, multiray spaces, and for $k = 3$ on trees and circles [12, 16, 21].

For randomized algorithms, the proof of a lower bound of k from the deterministic case does not work anymore. Instead, a lower bound of $\Omega(\log k)$ has been known for a long time. Let $H_k := \sum_{j=1}^{k}(1/j)$ denote the *k-th harmonic number*. On a clique of $k+1$ vertices, no deterministic algorithm is better than H_k-competitive in expectation on the instance distribution that requests any vertex except the one just requested with the same probability of $1/k$. *Yao's principle* [25] now transforms this lower bound for deterministic algorithms on an input distribution into a lower bound for randomized algorithms on fixed instances [6], [17, Thm. 2.11]. This has remained the best lower bound for multiple decades. (A series of results [3–5] showed a lower bound that is only $\Omega(\log k/(\log\log k))$ instead of $\Omega(\log k)$ but in exchange works on any metric space of at least $k+1$ points.) This led to the following folklore conjecture [19, Conj. 2].

▶ **Conjecture 2** (Randomized *k*-Server Conjecture)**.** *For any $k \geq 1$, there is an $\mathcal{O}(\log k)$-competitive randomized algorithm for the k-server problem in the distance model.*

Table 1 Overview of the best known upper and lower bounds (in the upper and lower row, respectively) on the competitive ratio for the k-server problem in the distance and time model, for deterministic algorithms on the line and general metric spaces, and additionally for randomized algorithms on general metric spaces. Note that all lower bounds on general graphs, apart from the deterministic lower bound of k in the distance model, are existential, i.e., they hold on specially constructed graphs.

Distance model			Time model		
Deterministic		Randomized	Deterministic		Randomized
Line	General	General	Line	General	General
k [9]	$2k-1$ [20]	$2k-1$ [20]	$(k^2+k)/2$ [22]	$2k^2-k$ [20, 22]	$2k^2-k$ [20, 22]
k [23]	k [23]	$\Omega((\log k)^2)$ [7]	$k+1$ [Th. 6]	$2k-1$ [Th. 12]	$k+H_k-1$ [Th. 16]

This conjecture was refuted in 2023 by Bubeck et al. [7], who were able to slightly lift the lower bound to $\Omega((\log k)^2)$ for one specific metric space. Note, however, the remaining exponential gap between this improved lower bound and the best known upper bound for randomized algorithms, which is still $2k-1$ and attained by the deterministic WFA.

A remarkable positive result by Bansal et al. [1] is a randomized algorithm that is, on metric spaces with N points, $\mathcal{O}((\log k)^2(\log N)^3 \log\log N)$-competitive in expectation, and thus even with high probability by a general result by Komm et al. [18]. Their algorithm thus obtains a competitive ratio that is polylogarithmic in k whenever N is polynomial in k. A more detailed discussion of the results up until 2009 is found in a survey by Koutsoupias [19].

The most relevant results for the k-server problem in the distance model on general metric spaces can thus be summarized as follows (see also Table 1): The best known upper bound on the competitive ratio is $2k-1$ (for both deterministic and randomized algorithms) and attained by the work function algorithm WFA [20]. The best known lower bound on the competitive ratio is $\Omega((\log k)^2)$ for randomized algorithms [7], and k for deterministic algorithms [23]. A matching deterministic upper bound of k is attained on the line by the double coverage algorithm DC [9], and on trees by the generalized variant DC-TREE [10].

The Time Model

Almost no research has been published on the time model, which allows us to describe all existing results in what follows.

Clearly, changing from the distance model to the time model helps the algorithm, because it can now move servers synchronously at no additional cost. However, this does not imply an improved competitive ratio. Since the advantages of the time model can be utilized by both the online algorithm and the offline solution it competes with, the ratio could, a priori, improve, stay the same, or get worse, and in different ways for different metric spaces.

A first result distinguishing the two variants was given by Koutsoupias and Taylor [22]. As they remark [22, Sect. 5], any upper bound for the distance model implies a k times larger upper bound for the time model: On the one hand, any algorithm for the distance model also works for the time model with the same or even lower cost. On the other hand, the optimal algorithm against which the online algorithm competes saves at most a factor of k in the time model compared to its cost in the distance model. Overall, the competitive ratio increases by at most a factor of k when an algorithm for the distance model is used in the time model. The $(2k-1)$-competitive WFA thus implies an upper bound of $2k^2-k$. For trees it immediately follows that DC-TREE, which is k-competitive in the distance model, is

k^2-competitive in the time model. Koutsoupias and Taylor [22, Thm. 3] lowered this bound to $k(k+1)/2$ with a straightforward adjustment of the potential function used in the analysis of DC-TREE; note that both bounds are in $\mathcal{O}(k^2)$. They concluded with a lower bound of 3 for $k = 2$ servers on the real line [22, Thm. 4], but provided no results for $k \geq 3$.

1.2 Contribution

The goal of this paper is to popularize the time-optimal k-server problem and establish it as a research object that is well worth the community's attention. We believe the k-server problem in the time model to be natural and well motivated. We also hope that the insights gained by analyzing the time model might shed further light on the long-standing open questions about the classical problem.

Our main contribution are lower bounds for the time-optimal k-server problem that are at least as large as and often larger than the best known lower bounds for the classical variant. For deterministic algorithms, we achieve the lower bound of k on all unweighted graphs with at least $k+1$ vertices (which excludes only cases where the competitive ratio is trivially 1) in Lemma 5. We then go above this, albeit just barely, with a lower bound of $k+1$ for a large class of metric spaces that includes all Euclidean spaces, infinite grids, and large cycles (Theorem 6). We then construct a specific metric space (superexponentially large in k, but finite and of diameter only 3) for which we can prove a lower bound of $2k-1$, which is exactly the best known upper bound for the classical k-server problem (Theorem 12).

Our main result for randomized algorithms is a lower bound on the expected competitive ratio for the time-optimal k-server problem (Theorem 16). The bound is $k + H_k - 1$, and thus exponentially larger than the recently proven polylogarithmic lower bound in the distance model [7].

Despite the significantly raised lower bounds, a gap still remains with a linear factor between the upper and lower bounds.

2 Preliminaries

For any nonnegative integer n, we use the notation $[n] \coloneqq \{1, \dots, n\}$. We denote by $H_k \coloneqq \sum_{j=1}^{k} (1/j)$ the k-th harmonic number.

Online Algorithms and Competitive Analysis

An online algorithm receives an instance I consisting of n *requests*, $I = (x_1, \dots, x_n)$, which are presented to ALG in n consecutive time steps; n is by default not known to ALG a priori. Whenever a request x_i is presented, ALG has to provide an *answer* y_i to x_i, where y_i depends only on the prefix $x_1, \dots, x_i$ of already presented requests. The full answer sequence $\text{ALG}(I) = (y_1, \dots, y_n)$ is the solution computed by ALG on I. Any solution $\text{ALG}(I)$ to I is assigned a *cost* denoted by $\text{cost}(I, \text{ALG}(I))$. An *optimal solution* for I, denoted by $\text{OPT}(I)$, is a solution with minimal cost across all solutions to I. Note that $\text{OPT}(I)$ and $\text{cost}(I, \text{OPT}(I))$ can generally be computed only once I is known.

The performance of an online algorithm ALG is measured in terms of its competitive ratio. ALG is called *c-competitive* if there is a constant α so that, for every instance I, it holds that $\text{cost}(I, \text{ALG}(I)) \leq c \cdot \text{cost}(I, \text{OPT}(I)) + \alpha$. Similarly, a randomized online algorithm RAND is said to be *c-competitive in expectation* or have an *expected competitive ratio of c* if there is a constant α so that, for every instance I, it holds that $\mathbb{E}[\text{cost}(I, \text{RAND}(I))] \leq c \cdot \text{cost}(I, \text{OPT}(I)) + \alpha$.

The k-Server Problem

The intuition behind the k-server problem is best described by a scenario where we are given a metric space and k *servers*, each occupying one point in the space. A request x is any point of the space, and an answer must specify at least one server that is moved to x in response. However, it is possible to move other servers as well. A request is revealed once the previous one has been served. The goal is to minimize either the total distance traveled after answering all requests (the distance model) or the overall time spent (the time model).

Formally, let $\mathcal{M} = (M, d)$ be any metric space. (In particular, the distance function $d\colon M^2 \to \mathbb{R}$ is zero on the pairs (x, x) for $x \in M$, otherwise positive, symmetric, and satisfies the triangle inequality, i.e., $\forall x, y, z \in M\colon d(x, z) \leq d(x, y) + d(y, z)$.) Any finite metric space can be described as the complete undirected graph whose vertices are M with an edge weight function $\{u, v\} \mapsto d(u, v)$. Conversely, we can interpret any finite, connected, weighted, and undirected graph as the metric space whose point set is the vertex set and whose distance function maps two points to the length of a shortest path between them.

In particular, any finite, connected, unweighted, and undirected graph also describes a metric space in the same way. From now on we always assume graphs in this paper to be finite, connected, undirected, and unweighted. Other important metric spaces include the real line, the Euclidean plane, and generally the Euclidean space of any given dimension.

For any positive integer k and metric space $\mathcal{M} = (M, d)$, we call a map $C\colon [k] \to M$ a *k-configuration*. In the context of the k-server problem we may describe a k-configuration by saying that server s_i is at point $C(i)$ for any $i \in [k]$. We say that a k-configuration C *covers* or *occupies* a point $p \in M$ with server s_i if $C(i) = p$ for some $i \in [k]$. For a sequence of k-configurations $C_0, \ldots, C_n$, we say, for any $i \in [k]$ and $j \in [n]$, that server s_i moves from $C_{j-1}(i)$ to $C_j(i)$ in step j. Whenever the value of k is clear from the context, which is the case for the remainder of this paper, we omit the k and simply say configuration instead of k-configuration. An *instance* of the k-server problem on $\mathcal{M}$ of length n is a sequence $(r_1, \ldots, r_n)$ of n points in M and an initial configuration $C_0\colon [k] \to M$. A *solution* is a sequence of configurations $(C_1, \ldots, C_n)$ such that C_j covers r_j for all $j \in [n]$. The *cost* of such a solution is $\sum_{i=j}^{n} \sum_{i=1}^{k} d(C_{j-1}(i), C_j(i))$ in the distance model and $\sum_{j=1}^{n} \max\{d(C_{j-1}(i), C_j(i)) \mid i \in [k]\}$ in the time model.

Intricacies of the Time Model

In the distance model, any algorithm can without loss of generality be transformed into one that is *lazy*, i.e., moves only one server at a time. Hence it is common and convenient to argue only about lazy algorithms. In the time model, this simplification is no longer justified; we also need to consider movements of the servers other than the one serving a request. This leads to intricacies peculiar to the time model, which we briefly discuss here.

Consider a graph with integer weights. Suppose that a server incident to an edge of unit length moves across this edge to serve a request at the other end. Suppose that another server is positioned at a vertex incident to a longer edge of length 2. Depending on the situation to be modeled, one might want to allow or disallow a synchronous movement by the second server to the midpoint of the longer edge. In the default formulation, servers can only traverse edges completely; partial movements are impossible. This implies the following perhaps unintuitive behavior: Even if a server that traversed a unit edge to serve a request subsequently moves back to serve the next request, traveling a length of 2 in two steps, it is impossible for another server to traverse an edge of length 2 at the same time without extra cost. Even if the servers move synchronously as far as possible, the cost is at least 3. This

problem does not occur in unweighted graphs, however. Indeed, if it is desired to enable the servers to traverse a long edge over multiple steps, this is easily achieved by transforming a given finite graph with rational weights into an unweighted one by subdividing the edges into segments of a length that divides all occurring weights.

All of our results apply to both variants since Robin (see Definition 3) works also on weighted graphs, and the graphs constructed for our lower bounds are all unweighted.

3 Results

We now present our results for the time-optimal k-server problem: an observation yielding an upper bound in Subsection 3.1, the lower bounds for deterministic algorithms in Subsection 3.2, and a lower bound for randomized algorithms in Subsection 3.3. Some proofs are summarized for our main theorems and removed for more straightforward results and corollaries. They can be found (in more detail) in the full version of the paper [15].

3.1 Upper Bounds

We start with some observations on a simple algorithm.

▶ **Definition 3** (Algorithm Robin)**.** *Label the k servers $s_1, \dots, s_k$ in arbitrary order. If a requested point is already covered by a server, then* Robin *does not change its configuration. The m-th request that requires* Robin *to move a server is served by s_i with $i \equiv_k m$, while all other servers stay idle. In other words,* Robin *moves a server only when it is necessary, only one server at a time, the first k movements are by $s_1, \dots, s_k$ in this order, and then it repeats cyclically.*

Note that Robin never moves servers synchronously, thus the cost is the same in the time model and the distance model. Essentially, Robin is a *marking algorithm* [6,17]. Koutsoupias and Taylor [22] already remarked that Robin is k-competitive on uniform metrics. It is also not hard to see that this observation can be extended to metrics of bounded *aspect ratio*, where the aspect ratio of a metric space is its diameter divided by the minimal distance between distinct points.

▶ **Observation 4.** *On metric spaces with aspect ratio A,* Robin *is Ak-competitive.*

3.2 Lower Bounds for Deterministic Algorithms

In this section, we provide lower bounds for deterministic algorithms on various metric spaces.

Universal Lower Bound of k on Unweighted Graphs

We begin with a bound of k for all unweighted graphs on which the problem is nontrivial.

▶ **Lemma 5.** *Let k be a positive integer. No algorithm for the time-optimal k-server problem can be better than k-competitive on graphs with more than k vertices.*

Proof sketch. We choose any connected set S of $k+1$ vertices of G and consider the metric subspace described by the induced subgraph $G[S]$. The algorithm is given an instance that always requests a point not occupied by any of its servers. We note that any configuration where all k servers occupy k distinct locations can be reached from any other such configuration at cost at most 1, by moving servers along a path connecting two unoccupied points. This allows an optimal offline algorithm to serve any k consecutive requests at cost 1. ◀

Note that, in contrast to the corresponding result for the distance model, it is unclear whether this result can be extended to arbitrary metric spaces with more than k points.

Lower Bound of $k + 1$ on the Line

We now consider a special type of metric space that has received a lot of attention (see Subsection 1.1) for the classical k-server problem: the line. As mentioned above, there are k-competitive algorithms for k-server in the distance model on the real line, the rational line, and the integer line. The following theorem shows that the time-optimal k-server problem is strictly harder in all of these cases.

▶ **Theorem 6.** *For any integer $k \geq 2$, no algorithm for the time-optimal k-server problem has a competitive ratio better than $k + 1$ on the line (real, rational, or integer), on any finite unweighted cycle with an even number of at least $2k + 6$ vertices, or on the continuous one-dimensional sphere $\mathbb{S}^1$.*

Proof. Let ALG be any online algorithm for the given metric space. For a finite unweighted cycle with an even number of $N \geq 2k + 6$ vertices, label the vertices counterclockwise by $0, 1, \ldots, N - 1$. For the sphere $\mathbb{S}^1$, choose $N = 2k + 6$ points evenly spaced along this metric space, label them $0, 1, \ldots, N - 1$ counterclockwise, and re-scale the space such that the distance between consecutive points is 1. For simplicity, we refer to the labeled points as integers and say that an integer *neighbors* another one if they are at distance 1 from each other. In all cases, every even integer neighbors two odd ones and each odd one two even ones. We will construct an instance phase by phase such that there is an optimal algorithm OPT satisfying the following invariant.

▶ **Invariant 7.** *At the beginning of each phase, the k servers of* OPT *occupy k distinct even integers or k distinct odd integers. Moreover, at least two of these integers are consecutive (in the sense of being separated by a distance of exactly 2).*

Without loss of generality, we assume for the following description that the integers occupied by OPT at the start of the phase are all even by choosing an appropriate initial configuration and shifting the labels of the integers by one after each phase. Any phase will consist of potentially repeated requests for k distinct odd integers, among which at least two are consecutive. Moreover, the requests will be chosen such that OPT can cover all of them by moving all of its k servers simultaneously, each by a distance 1 and thus at total cost 1, at the beginning of the phase and with no movements afterwards. Any phase constructed with these properties preserves the invariant for the next phase, allowing us to iterate the process to construct an arbitrarily long instance of arbitrarily high optimal cost.

Since OPT does not move its servers anymore after its phase-initial synchronous movements, we can request any point covered by OPT as many times as we would like without increasing the cost of OPT or changing OPT's configuration at the end of the phase. We can thus enforce the following invariant for ALG without loss of generality.

▷ Claim 8. Once ALG has served a request at some point during some phase, it will always have a server at this point during all further requests of this phase.

Proof of claim. Assume that ALG makes a move such that some integer previously requested in the current phase is no longer covered. Then the constructed instance could be extended by immediately requesting this integer again after this move, at no additional cost to OPT. This can be continued until all the integers previously requested in the current phase are covered again by ALG. ◁

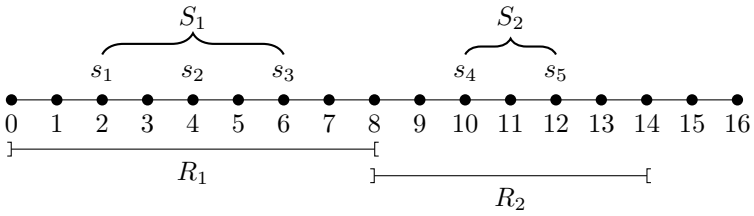


Figure 1 Example of the subdivision of phase-initial server positions and the corresponding ranges used in the proof of Theorem 6. (Note that the figure does not show the entire space, which is a cycle or the infinite line, but only a segment.)

Using this invariant of ALG, we can now also assume the following without loss of generality.

▷ Claim 9. In each phase, k distinct odd integers are requested, each exactly once.

In summary, we can assume the following properties for each phase.

1. At the beginning of the phase, the servers of ALG occupy the same positions as the servers of OPT, namely k *distinct* even integers, at least two of which are consecutive (in the sense of being separated by a distance of 2).
2. During the phase, k distinct odd integers are requested one by one. Each such integer must, once requested, be covered by ALG during all subsequent requests of the phase.
3. Moreover, there is a bipartite matching between the k odd requested points and the k phase-initially covered even integers where each edge of the matching has weight exactly 1.
4. Finally, OPT moves its servers along this matching at cost 1 when the first request of the phase appears and does not move its servers anymore for the rest of the phase.

We now describe how the k requests for any given phase are chosen depending on ALG's behavior in a way that guarantees Invariant 7 to hold for the following phase.

Consider any phase. Partition the phase-initial server positions (which are the same for OPT and ALG) into $\ell \geq 1$ maximal, pairwise disjoint, nonempty sets $S_1, \dots, S_\ell$ of consecutive even integers. We define $k_i := |S_i|$ and note that $\sum_{i=1}^{\ell} k_i = k$. Given a metric space $\mathcal{M} = (M, d)$, a point $c \in M$, and radius $\rho \geq 0$, we call $D_\rho(c) := \{p \in M \mid d(p,c) < \rho\}$ the *open ball* and $D_\rho[c] := \{p \in M \mid d(p,c) \leq \rho\}$ the *closed ball*. For any $m \in [\ell]$, we call $R_m := \bigcup_{i \in S_m} D_2(i)$ the *range* of S_m; Figure 1 shows an example for $k = 5$.

Note that the range R_m contains exactly $k_m + 1$ odd integers. Exactly k_m of them are requested during a phase in the instance family described below. This guarantees that OPT can indeed move all its servers from their phase-initial positions to the points request during this phase at cost 1, and then keep them there for the remainder of the phase, which proves the first part of Invariant 7. We call the range R_m *saturated* if the instance has already requested k_m out of the $k_m + 1$ odd integers of R_m during the current phase, and *unsaturated* otherwise. Recall that once an integer is requested, a server occupies it during all remaining requests of the phase; thus any saturated R_m contains at least k_m servers. A phase ends when all ranges are saturated.

Since there are two consecutive even integers in the phase-initial configuration of OPT, by renaming we can assume without loss of generality that $k_1 \geq 2$ and that $S_1 = \{2, 4, \dots, 2k_1\}$. Since R_1 contains exactly $k_1 + 1 \geq 3$ consecutive odd integers and k_1 of them are requested during the phase, at least two consecutive odd integers are requested. This proves the second part of Invariant 7.

We now describe what the requests in the considered phase depend on. For this, we will call a point *distant* if and only if it has distance at least 1 from all current servers, i.e., if it lies outside of all open unit balls around the current server locations. The first request is 3, which is indeed a distant point since all servers are phase-initially at even integers. From then on, a further request is chosen according to the following *selection procedure* until $k-1$ odd integers have been requested during the phase or the procedure becomes impossible: Request an arbitrary distant odd integer not previously requested during the phase from any unsaturated range R_m except for the integer 1. The sequence of such requests ends only when k_m requests have been made in each range R_m with $1 \neq m \in [\ell]$ and $k_1 - 1$ requests have been made in R_1, or if no such request is possible anymore. We will show that once this process ends, there must be a point in R_1 that no server can reach with cost less than 2 using the following simple claim.

▷ Claim 10. If a range R_m for any $m \in [\ell]$ contains at most k_m servers, it also contains a distant odd integer. If the *truncated range* $R'_1 \coloneqq R_1 \setminus (0, 4]$ contains at most $k_1 - 2$ servers, it contains a distant odd integer.

Proof of claim. A unit ball can contain at most one odd integer. Moreover, unit balls around points outside of a range R_m cannot contain odd integers inside this range, and analogously for the truncated range R'_1. Hence, the range R_m for any $m \in [\ell]$ containing at most k_m servers means that at most k_m out of the $k_m + 1$ odd integers in it are not distant and thus that there is a distant odd integer in R_m. Analogously, the truncated range R'_1 containing at most $k_1 - 2$ servers means that at most $k_1 - 2$ out of the $k_1 - 1$ odd integers in it are not distant, and thus that there is a distant odd integer in R'_1. ◁

By this claim, if the selection procedure ends, any range R_m with $1 \neq m \in [\ell]$ must contain at least k_m servers, whether or not it is saturated, and R'_1 must contain at least $k_1 - 2$ servers. We consider two different cases.

- *Case 1: R'_1 contains at least $k_1 - 1$ servers.* In this case, R_1 contains at least k_1 servers (including the server positioned on 3). This means that all ranges R_m with $1 \neq m \in [\ell]$ must contain exactly k_m servers and must therefore be saturated. Thus, these servers all occupy odd integers and have a distance of at least 2 to the odd integer 1. The same holds for any servers in R'_1.
- *Case 2: R'_1 contains exactly $k_1 - 2$ servers.* In this case, R'_1 contains a distant point. Since the selection procedure ended, a total of $k_1 - 1$ requests must have already been made to R_1 and served. Thus, the $k_1 - 2$ servers in R'_1 occupy distinct odd integers and cannot reach any other odd integer in R_1 with cost less than 2. Of the other ranges R_m with $1 \neq m \in [\ell]$, all but possibly one must be saturated, since they contain exactly k_m servers. Therefore, these servers occupy odd integers and cannot reach odd integers in R_1 with cost less than 2. Finally, one range R_m with $1 \neq m \in [\ell]$ may contain $k_m + 1$ servers, or a single server may occupy a point outside a range. Such a server can only reach either the odd integer 1 or the odd integer $2k_1 - 1$ in R_1 with cost less than 2. This is clear if the metric space is a line.

  If it is a cycle or the continuous sphere, the two ranges would have to meet at both ends. However, this would mean that they contain all odd integers in the metric space, of which there are at least $k + 3$, while they contain a maximum of $(k_1 + 1) + (k_m + 1) \leq k + 2$ odd integers. Thus, since there are two odd integers in R_1 that have not been requested yet, one of them cannot be reached by any server with cost less than 2.

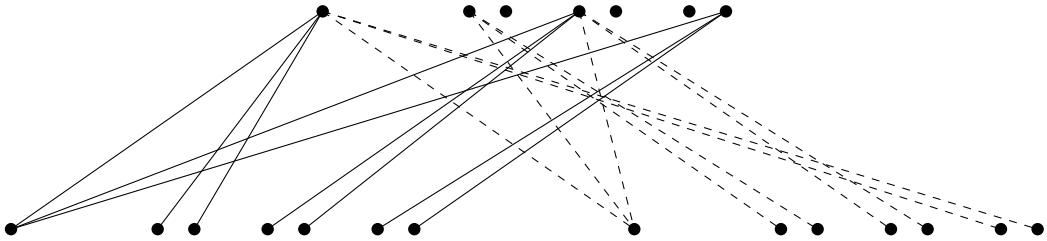


Figure 2 A small part of the construction used to prove Theorem 12 for $k = 3$: one of the $k(k-1)^{k-1} = 12$ blocks of one layer at the top (with the hub point on the left and the fringe points in k groups of $k-1$ to the right), two blocks of the next layer at the bottom left and bottom right, and the edges induced by the corresponding choices of $k-1$ groups and one fringe point per chosen group in the block above; with dashed edges for the choice represented by the bottom right block.

We thus have, in both cases, an odd integer in R_1 that is at distance at least 2 from all servers. Such an integer is requested next, forcing ALG to incur a cost of at least 2 when serving it. From then on, additional distant points in unsaturated ranges R_m with $m \in [\ell]$ (including R_1) are requested. Such points exist by Claim 10. Once k distinct odd integers have been requested, all ranges are saturated and the phase ends.

Recall that the first request was to the odd integer 3. Since R_1 contains $k_1 + 1$ odd integers, k_1 of which have been requested, at least one of the odd integers 1 or 5 must have been requested, and thus indeed at least two consecutive odd integers. This means that the final configuration of the phase is such that Invariant 7 is satisfied for an immediately following phase.

We can thus iterate the process and construct arbitrarily many new phases such that for each phase ALG incurs a cost of at least $k + 1$ while OPT has a cost of exactly 1. ◀

Note that Theorem 6 generalizes the lower bound of 3 given by Koutsoupias and Taylor for the special case of $k = 2$ servers on the real line [22, Thm. 4]. In contrast to the distance model, lower bounds in the time model do not trivially extend to arbitrary metric superspaces. However, this particular result can be extended to other metric spaces, e.g., the Euclidean plane.

▶ **Corollary 11.** *No algorithm can solve the time-optimal k-server problem with a competitive ratio better than $k + 1$ in the Euclidean space of any positive dimension.*

Existential Lower Bound of $2k - 1$ on General Graphs

The most celebrated result for the k-server problem is Koutsoupias and Papadimitriou's proof [20] showing WFA to be $(2k-1)$-competitive in the distance model. Intriguingly, $2k-1$ is also our best *lower* bound for the time model.

▶ **Theorem 12.** *For any $k \geq 1$, there is a finite metric space on which no online algorithm for the time-optimal k-server problem has a competitive ratio better than $2k - 1$.*

Proof. We first give the general idea of the proof. Consider a star with k rays of length 2, and assume that there is a server at each midpoint of a ray. The center of the star is requested first. Any algorithm must move one server there at cost 1. With one of the k servers at the center, one ray no longer contains a server. Its outer point is requested next, causing a cost of at least 2. This process is then repeated $k-1$ times, always requesting the outer point of an unoccupied ray, for a total cost of $2k - 1$.

The main challenge lies in simulating this instance while making the process repeatable. To this end, we construct a metric space given by a finite, unweighted graph.

Description of the metric space. The metric space is a tripartite graph $G = (V, E)$, i.e., its vertices can be partitioned into three disjoint *layers* $L_1 \cup L_2 \cup L_3 = V$ such that the edges can be partitioned as $E_1 \cup E_2 \cup E_3 = E$ with $E_1 \subseteq L_1 \times L_2$, $E_2 \subseteq L_2 \times L_3$, and $E_3 \subseteq L_3 \times L_1$.

On the instances we describe later, the three layers will be used cyclically by any optimal solution: its servers will all move synchronously first from L_1 to L_2, then to L_3, then back to L_1, and so on. Moreover, the three subgraphs $(L_1 \cup L_2, E_1)$, $(L_2 \cup L_3, E_2)$, and $(L_3 \cup L_1, E_3)$ are all isomorphic. In particular, the layers all have the same size. Each layer is partitioned into $B \coloneqq k(k-1)^{k-1}$ uniformly sized *blocks*. Each block in turn consists of one special vertex called its *hub* and $k(k-1)$ vertices that we call its *fringe points*, grouped into k groups each of size $k-1$. We identify the blocks with the integers $[B]$, and similarly denote the groups of each block by $[k]$ and the points of a group by $[k-1]$. We denote the hub of block b in layer L_ℓ by $h(\ell, b)$ and the fringe point n in group g of the same block by $f(\ell, b, g, n)$. Thus, the vertex set is given by

$$V \coloneqq \bigcup_{\ell=1}^{3} \bigcup_{b=1}^{B} \left(\{h(\ell, b)\} \cup \bigcup_{g=1}^{k} \bigcup_{n=1}^{k-1} \{f(\ell, b, g, n)\} \right) .$$

For notational convenience we define $L_4 \coloneqq L_1$ and $L_0 \coloneqq L_3$. To describe the edges of G, we first highlight some uniform aspects of the construction. For each $\ell \in [3]$, the edges E_ℓ between L_ℓ and $L_{\ell+1}$ are constructed identically.

Moreover, the edges from any layer to the next are *block-uniform* in the following sense: for any $\ell \in [3]$, $g \in [k]$, and $n \in [N]$, the neighborhood in $L_{\ell+1}$ is the same for every fringe point $f(\ell, b, g, n)$ and for any hub $h(\ell, b)$ independent of $b \in [B]$.

Now consider any block b of L_ℓ. To each possible choice of $k-1$ fringe points from $k-1$ distinct groups, we assign one block $c \in [B]$ from $L_{\ell+1}$ by an arbitrary but fixed bijection. Such a bijection exists, because there are k possibilities to choose $k-1$ fringe groups and $(k-1)^{k-1}$ ways to choose one fringe point from each of these groups, which means that the total number of possible choices is exactly B, the number of blocks in $L_{\ell+1}$.

Fix such a choice and the corresponding block c in $L_{\ell+1}$. Each of the $k-1$ chosen fringe points, as well as the hub of b, is assigned injectively to one of the k groups of fringe points of c and then made adjacent to all $k-1$ points of the assigned group. For concreteness, let the chosen fringe point of group g in b be assigned to group g in block c, while the hub is assigned to the remaining group of block c that is not represented in the $k-1$ chosen ones. Finally, all $k-1$ chosen fringe points and the hub of b are made adjacent to the hub of c in $L_{\ell+1}$.

More formally, identify the possible choices of $k-1$ fringe points in a block of L_ℓ with $[B]$. For each choice $c \in [B]$, denote by $g_c \in [k]$ the unique index of the group without a chosen point. For any $g \in [k]$ with $g \neq g_c$, denote by $n_c^g \in [k-1]$ the index of the unique point chosen from group g. This means that the fringe points in choice c are exactly $f(\ell, b, g, n_c^g)$ for $g \in [k]$ and $g \neq g_c$. Then we have the following edges between block b, which is in L_ℓ, and $L_{\ell+1}$:

$$\begin{aligned} E(\ell, b) \coloneqq & \bigcup_{c=1}^{B} \{\{h(\ell, b), h(\ell+1, c)\}\} \\ & \cup \bigcup_{c=1}^{B} \bigcup_{n=1}^{k-1} \{\{h(\ell, b), f(\ell+1, c, g_c, n)\}\} \end{aligned}$$

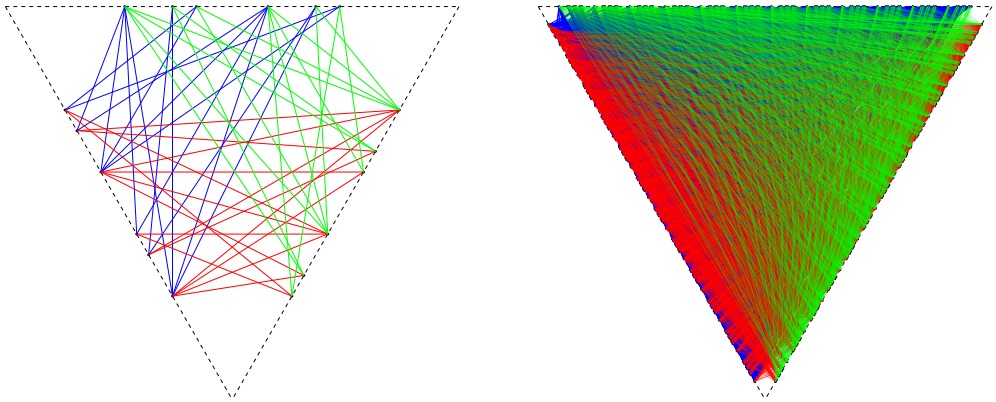


Figure 3 The graph describing the metric space used to prove Theorem 12 for $k = 2$ (left) and $k = 3$ (right). The layers L_1, L_2, and L_3 are arranged counterclockwise with L_1 at the top. Edges from L_1 to L_2 are shown in blue, those from L_2 to L_3 in red, and those from L_3 to L_1 in green.

$$\cup \bigcup_{c=1}^{B} \bigcup_{\substack{g\in[k]\\ g\neq g_c}} \{\{f(\ell, b, g, n_c^g), h(\ell+1, c)\}\}$$
$$\cup \bigcup_{c=1}^{B} \bigcup_{\substack{g\in[k]\\ g\neq g_c}} \bigcup_{n=1}^{k-1} \{\{f(\ell, b, g, n_c^g), f(\ell+1, c, g, n)\}\} .$$

The full edge set is then

$$E \coloneqq \bigcup_{\ell=1}^{3} \bigcup_{b=1}^{B} E(\ell, b) .$$

A visual representation of the graph $G = (V, E)$ for $k = 2$ is shown in Figure 3. For $k = 3$, an example of one block in L_1 and two blocks in L_2 is given in Figure 2, while the full graph is shown in Figure 3. The following claim summarizes some important properties of G. It follows directly from the construction of the edges.

▷ Claim 13. Fix any block b in L_ℓ. We have the following properties:

- **(i)** Any vertex in $L_{\ell-1}$ is adjacent to vertices of at most one fringe group of block b.
- **(ii)** Any fringe point in $L_{\ell+1}$ is adjacent to at most one fringe point of block b.
- **(iii)** Any hub point in $L_{\ell+1}$ is adjacent to exactly one vertex for each of exactly $k-1$ of the k fringe groups of block b.

Description of the instances. We now show that no online algorithm can be better than $(2k-1)$-competitive on the metric space described by G.

Let ALG be any deterministic online algorithm. We describe an instance consisting of distinct phases. In each phase, exactly k distinct points, all from the same layer, are requested, potentially with many repetitions. The instances are designed such that OPT can move all its k servers by a distance of 1 at the beginning of each phase and then serve all requests within this phase at no additional cost, while ALG has cost at least $2k-1$.

At the start of each phase, the servers of OPT are located in a single block b of L_ℓ and more specifically at one of its hub points and $k-1$ points chosen from $k-1$ distinct fringe groups in it.

The phase consists of vertices from the block c in $L_{\ell+1}$ corresponding to the choice of fringe points in block b that form the configuration of OPT at the start of the phase. Specifically, it consists of the hub of block c and $k-1$ fringe points chosen from distinct groups. This means that the process can be repeated since the final configuration is equivalent to the starting position. It also means that OPT can reach the configuration at cost 1: it can move $k-1$ of its servers in block b to fringe points in block c in the unique group they are adjacent to, and move the final server to the hub of block c.

We can assume that within each phase, ALG keeps a server on each previously requested point. Otherwise, the instance will simply request that point again, at no additional cost to OPT.

The instance starts the phase with a request to the hub of block c. ALG must now move a server to that vertex and thus incurs cost 1. Assume that $s \in [k-1]$ fringe points in block c have already been requested by the instance in this phase. We claim that there is at least one fringe point n in block c such that

(a) no vertex from the group in block c containing n was requested in this phase before, and

(b) no server of ALG can reach n with cost less than 2.

By assumption, $s+1$ servers of ALG already occupy requested points in block c. These points are not adjacent to any other point in L_2, and thus in particular not to any point in block c. There are $k-s$ groups in block c that are not yet represented by a request in this phase. Since there are only $k-s-1$ remaining servers, by Claim 13(i) and (ii), there is at least one group in block c such that no fringe point in this group is adjacent to a server that is in $L_{\ell-1}$ or on a fringe point in $L_{\ell+1}$. Within this group, there are $k-1$ points. Any server of ALG that can reach such a point with cost at most 1 is either on that point (in which case it cannot reach any other point in block c with cost at most 1) or on a hub in $L_{\ell+1}$ (in which case, by Claim 13(iii), the same holds).

Since there are $k-s-1$ remaining servers, this means that such a point n exists, unless $s=0$, i.e., if the only point requested in this phase has been the hub, and all servers of ALG occupy hubs in $L_{\ell+1}$ or fringe points in L_ℓ. In this case, by Claim 13(iii), a server on a hub in $L_{\ell+1}$ can reach at most $k-1$ fringe points with cost 1. A server on a fringe point in block c can only reach one such point. Therefore, the total number of fringe points in block c that can be reached in distance 1 by servers on such vertices is at most $(k-1)^2$. This is strictly smaller than the total number of $k \cdot (k-1)$ fringe points in block c; so again, a point n with the required properties must exist.

The instance now requests this point, which ALG must serve with cost at least 2. After $k-1$ such requests, the phase ends. ALG and OPT are in the promised final configuration, and the total cost of ALG in this phase is $1 + (k-1) \cdot 2 = 2k-1$. ◀

▶ **Observation 14.** *The diameter of the metric space used in the proof of Theorem 12 is* 3.

It follows by Observation 4 that there is a $3k$-competitive algorithm on this graph.

3.3 Lower Bound for Randomized Algorithms

In this section, we provide a lower bound for randomized algorithms. We first state Yao's principle for infinite minimization problems [17, Thm. 2.5]. A comprehensive explanation is found in the textbooks by Borodin and El-Yaniv [6], and Komm [17].

▶ **Fact 15** (Yao's Principle). *Let $\mathcal{I}_1, \mathcal{I}_2, \dots$ be an infinite sequence of sets of instances of a given (online) minimization problem. Let $A_1, A_2, \dots$ be a list of all deterministic online algorithms for this problem. Let $\Pr_i$ denote an adversarial probability distribution on the instances in $\mathcal{I}_i$, and let $\mathbb{E}_i$ denote the corresponding expected value function. If there is some constant $c \geq 1$ such that*

(i) $\dfrac{\min_j(\mathbb{E}_i[\mathrm{cost}(A_j(\mathcal{I}_i))])}{\mathbb{E}_i[\mathrm{cost}(\mathrm{OPT}(\mathcal{I}_i))]} \geq c$ *for every positive integer i, and*

(ii) $\lim_{i\to\infty} \mathbb{E}_i[\mathrm{cost}(\mathrm{OPT}(\mathcal{I}_i))] = \infty$,

then there is, for any given constant $\varepsilon > 0$, no randomized online algorithm for the given problem that is $(c-\varepsilon)$-competitive in expectation.

Existential Lower Bound of $k + H_k - 1$ on General Graphs

We now provide a lower bound of $k+\Omega(\log k)$ on the expected competitive ratio of randomized online algorithms. This is achieved on a significant extension of the metric space used in the proof of Theorem 12.

Note that it is easy to find a metric space on which no randomized algorithm can have a competitive ratio better than $\Omega(k)$. In fact, on uniform metric spaces with $N \gg k$, there is a lower bound of $k - o(1)$, which can be shown by requesting points uniformly at random and using Yao's principle. The value of Theorem 16 lies in showing a bound that is *strictly larger* than k, the best known lower bound and conjectured competitive ratio for deterministic algorithms in the classical distance model.

▶ **Theorem 16.** *Let any positive integer $k \geq 1$ be given. For any $\varepsilon > 0$, there is a finite metric space on which no randomized online algorithm for the time-optimal k-server problem has a better constant competitive ratio than $k + H_k - 1 - \varepsilon$.*

Proof sketch. The metric space is described by a graph $G = (V, E)$ that extends the construction used in the proof of Theorem 12. Instead of a single hub point $h(\ell, b)$, each block contains a *hub group* of N hub points $h(\ell, b, n)$ for some large number N. Each of the k fringe groups also contains N fringe points. Correspondingly, the number of blocks is $B = kN^{k-1}$.

We describe a distribution over hard instances on which no deterministic algorithm ALG can be better than $(k + H_k - 1 - \varepsilon)$-competitive, and then apply Yao's principle. The instances are again partitioned into phases, and in each phase, points are requested from the block corresponding to the positions of the servers of OPT, so that OPT has cost 1 per phase.

The instances start with a hub point chosen uniformly at random from this block. The expected cost ALG incurs serving this request is approximately 1 since there are $N \gg k$ possible choices. From then on, one of the $k+1$ groups (either the hub group or a fringe group) is chosen uniformly at random. If no point from that group has been requested yet in that phase, a point from the group is chosen uniformly at random and requested; otherwise, the previous point is requested again.

We then show that, if s distinct points with $1 \leq s < k$ have been requested, the expected cost of ALG is at least $(k+1-s)/(k+1)+1/(k+1)$, by using the equivalent of Claim 13. The expected number of requests until the next distinct point is requested is $(k+1)/(k+1-s)$, so the expected cost of ALG in that timeframe is at least $1+1/(k+1-s)$. Summing up over all s and adding the expected cost of 1 for the first request to a hub point, the expected cost of ALG is therefore

$$1 + \sum_{s=1}^{k-1} \left(1 + \frac{1}{k+1-s}\right) = k + H_k - 1\,. \qquad \blacktriangleleft$$

Note that it can be shown that the graph used in the proof of Theorem 16 has diameter 3, analogously to Observation 14. Thus there is also a deterministic algorithm with competitive ratio $3k$ on that metric space.

4 Conclusion

We hope to initiate a line of research that focuses on the time-optimal k-server problem. We started by proving a series of lower bounds, showing the time model to be harder than the distance model on various metric spaces, including simple cycles and all Euclidean spaces, which implies that the direct analogue of the k-server conjecture in the time model cannot be true. Our strongest deterministic lower bound matches – intriguingly – the best upper bound known for the classical distance-optimal k-server problem, which is attained by the deterministic work function algorithm WFA. A priori, it could thus be true that this algorithm is in fact exactly $(2k-1)$-competitive on general metric spaces in both models.

A natural next step is to find good performance guarantees for WFA in the time-optimal setting. Unfortunately, the analysis of Koutsoupias and Papadimitriou [20] that proved successful for the distance model resists any straightforward adaptation; multiple key concepts, such as the duality between the so-called minimizers and maximizers, do not translate well to the time model. Better upper bounds in the time model will thus probably provide us with several novel techniques.

Our preliminary attempts and experimentally gathered evidence indeed point to a subquadratic upper bound. We believe the lower bound of Theorem 12 to be tight, i.e., we conjecture that a competitive ratio of $2k-1$ is indeed attainable.

▶ **Conjecture 17.** *There is a deterministic algorithm for the time-optimal k-server problem with a competitive ratio of $2k-1$ on general metric spaces.*

For randomized algorithms, we analogously suspect the existence of an expected competitive ratio of c in the distance model to correspond to one of $c+k-1$ in the time model.

References

1. Nikhil Bansal, Niv Buchbinder, Aleksander Mądry, and Joseph (Seffi) Naor. A polylogarithmic-competitive algorithm for the k-server problem (extended abstract). In *Proceedings of the 52th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2011)*, pages 267–276, 2011. doi:10.1109/FOCS.2011.63.
2. Nikhil Bansal, Niv Buchbinder, and Joseph (Seffi) Naor. Metrical task systems and the k-server problem on HSTs. In *Proceedings of the 37th International Colloquium on Automata, Languages and Programming (ICALP 2010)*, volume 6198 of *LNCS*, pages 287–298, 2010. doi:10.1007/978-3-642-14165-2_25.
3. Yair Bartal, Béla Bollobás, and Manor Mendel. Ramsey-type theorems for metric spaces with applications to online problems. *Journal of Computer and System Sciences (JCSS)*, 72(5):890–921, 2006. doi:10.1016/J.JCSS.2005.05.008.
4. Yair Bartal, Nathan Linial, Manor Mendel, and Assaf Naor. On metric Ramsey-type phenomena. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC 2003)*, pages 463–472, 2003. doi:10.1145/780542.780610.
5. Avrim Blum, Howard J. Karloff, Yuval Rabani, and Michael E. Saks. A decomposition theorem and bounds for randomized server problems. In *33rd Annual Symposium on Foundations of Computer Science (FOCS 1992)*, pages 197–207, 1992. doi:10.1109/SFCS.1992.267772.
6. Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, Cambridge, 1998. doi:10.5555/290169.

7 Sébastien Bubeck, Christian Coester, and Yuval Rabani. The randomized k-server conjecture is false! In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC 2023)*, pages 581–594, 2023. doi:10.1145/3564246.3585132.

8 Niv Buchbinder and Joseph (Seffi) Naor. The design of competitive online algorithms via a primal-dual approach. *Foundations and Trends in Theoretical Computer Science*, 3(2–3), 2009. doi:10.1561/0400000024.

9 Marek Chrobak, Howard J. Karloff, Thomas H. Payne, and Sundar Vishwanathan. New results on server problems. *SIAM Journal on Discrete Mathematics*, 4(2):172–181, 1991. doi:10.1137/0404017.

10 Marek Chrobak and Lawrence L. Larmore. An optimal on-line algorithm for k-servers on trees. *SIAM Journal on Computing*, 20(1):144–148, 1991. doi:10.1137/0220008.

11 Marek Chrobak and Lawrence L. Larmore. The server problem and on-line games. In *On-Line Algorithms, Proceedings of a DIMACS Workshop*, volume 7, pages 11–64, 1991. doi:10.1090/DIMACS/007/02.

12 Christian Coester and Elias Koutsoupias. Towards the k-server conjecture: A unifying potential, pushing the frontier to the circle. In *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, volume 198 of *LIPIcs*, pages 57:1–57:20, 2021. doi:10.4230/LIPICS.ICALP.2021.57.

13 Amos Fiat, Yuval Rabani, and Yiftach Ravid. Competitive k-server algorithms (extended abstract). In *31st Annual Symposium on Foundations of Computer Science (FOCS 1990)*, LIPIcs, pages 454–463, 1990. doi:10.1109/FSCS.1990.89566.

14 Amos Fiat, Yuval Rabani, and Yiftach Ravid. Competitive k-server algorithms. *Journal of Computer and System Sciences (JCSS)*, 48(3):410–428, 1994. doi:10.1016/S0022-0000(05)80060-1.

15 Fabian Frei, Dennis Komm, Moritz Stocker, and Philip Whittington. Time-optimal k-server, 2025. doi:10.48550/arXiv.2503.05589.

16 Zhiyi Huang and Hanwen Zhang. Deterministic 3-server on a circle and the limitation of canonical potentials. *Theoretical Computer Science*, 1020:114844, 2024. doi:10.1016/j.tcs.2024.114844.

17 Dennis Komm. *An Introduction to Online Computation – Determinism, Randomization, Advice*. Springer, 2016. doi:10.1007/978-3-319-42749-2.

18 Dennis Komm, Rastislav Královič, Richard Královič, and Tobias Mömke. Randomized online algorithms with high probability guarantees. *Algorithmica*, 84(5):1357–1384, 2022. doi:10.1007/s00453-022-00925-z.

19 Elias Koutsoupias. The k-server problem. *Computer Science Review*, 3(2):105–118, 2009. doi:10.1016/j.cosrev.2009.04.002.

20 Elias Koutsoupias and Christos H. Papadimitriou. On the k-server conjecture. *Journal of the ACM*, 42(5):971–983, 1995. doi:10.1145/210118.210128.

21 Elias Koutsoupias and Christos H. Papadimitriou. The 2-evader problem. *Information Processing Letters*, 57(5):473–482, 1996. doi:10.1016/0020-0190(96)00010-5.

22 Elias Koutsoupias and David Scot Taylor. The CNN problem and other k-server variants. *Theoretical Computer Science (TCS)*, 324(2-3):347–359, 2004. doi:10.1016/J.TCS.2004.06.002.

23 Mark S. Manasse, Lyle A. McGeoch, and Daniel D. Sleator. Competitive algorithms for on-line problems. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing (STOC 1988)*, pages 322–333, 1988. doi:10.1145/62212.62243.

24 Mark S. Manasse, Lyle A. McGeoch, and Daniel Dominic Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11(2):208–230, 1990. doi:10.1016/0196-6774(90)90003-W.

25 Andrew C.-C. Yao. Probabilistic computations: Toward a unified measure of complexity (extended abstract). In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS 1977)*, pages 222–227, 1977. doi:10.1109/SFCS.1977.24.

Coloring Reconfiguration Under Color Swapping

Janosch Fuchs
IT Center, RWTH Aachen University, Germany

Rin Saito
Graduate School of Information Sciences, Tohoku University, Japan

Tatsuhiro Suga
Graduate School of Information Sciences, Tohoku University, Japan

Takahiro Suzuki
Graduate School of Information Sciences, Tohoku University, Japan

Yuma Tamura
Graduate School of Information Sciences, Tohoku University, Japan

Abstract

In the Coloring Reconfiguration problem, we are given two proper k-colorings of a graph and asked to decide whether one can be transformed into the other by repeatedly applying a specified recoloring rule, while maintaining a proper coloring throughout. For this problem, two recoloring rules have been widely studied: *single-vertex recoloring* and *Kempe chain recoloring*. In this paper, we introduce a new rule, called *color swapping*, where two adjacent vertices may exchange their colors, so that the resulting coloring remains proper, and study the computational complexity of the problem under this rule. We first establish a complexity dichotomy with respect to k: the problem is solvable in polynomial time for $k \leq 2$, and is PSPACE-complete for $k \geq 3$. We further show that the problem remains PSPACE-complete even on restricted graph classes, including bipartite graphs, split graphs, and planar graphs of bounded degree. In contrast, we present polynomial-time algorithms for several graph classes: for paths when $k = 3$, for split graphs when k is fixed, and for cographs when k is arbitrary.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Combinatorial reconfiguration, graph coloring, PSPACE-complete, graph algorithm

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.33

Related Version *Full Version*: https://arxiv.org/abs/2511.06473

Funding *Rin Saito*: Supported by JST SPRING Grant Number JPMJSP2114.
Yuma Tamura: Supported by JSPS KAKENHI Grant Number JP25K21148.

1 Introduction

1.1 Reconfiguring of Colorings

The field of *combinatorial reconfiguration* investigates reachability and connectivity in the solution space of combinatorial problems. A combinatorial reconfiguration problem is defined with respect to a combinatorial problem Π and a *reconfiguration rule* R, which specifies how one feasible solution of Π can be transformed into another. Given an instance of Π and two feasible solutions, the reconfiguration problem asks whether it is possible to transform one into the other through a sequence of solutions, each obtained by a single application of R, such that all intermediate solutions are also feasible. Combinatorial reconfiguration problems naturally arise in applications involving dynamic systems, where solutions must be updated while maintaining feasibility at every step. In addition, studying such problems provides deeper insights into the structural properties of the solution spaces of classical combinatorial problems. We refer the reader to the surveys by van den Heuvel [20] and Nishimura [31] for comprehensive overviews of this growing area.

© Janosch Fuchs, Rin Saito, Tatsuhiro Suga, Takahiro Suzuki, and Yuma Tamura;
licensed under Creative Commons License CC-BY 4.0

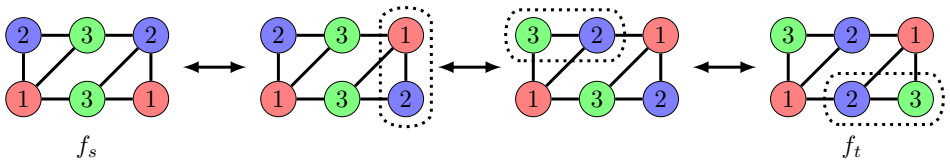


Figure 1 A reconfiguration sequence between two proper 3-colorings f_s and f_t under color swapping.

Among the various reconfiguration problems, one of the most fundamental and extensively studied is the reconfiguration of graph colorings. Let k be a positive integer. For a fixed reconfiguration rule R, the Coloring Reconfiguration problem under R is defined as follows: given a k-colorable graph G and two proper k-colorings f and f', determine whether there exists a sequence of proper k-colorings of G starting at f and ending at f', where each coloring in the sequence is obtained from the previous one by a single application of R. The k-Coloring Reconfiguration is a special case where k is fixed. The computational complexity of k-Coloring Reconfiguration has been the subject of extensive algorithmic study; see Section 3 of the survey by Mynhardt and Nasserasr [30].

Two reconfiguration rules have been widely studied for k-Coloring Reconfiguration: *single-vertex recoloring* and *Kempe chain recoloring*. In the *single-vertex recoloring* rule, a new proper k-coloring is obtained by recoloring a single vertex so that the resulting k-coloring remains proper. Under this rule, k-Coloring Reconfiguration is solvable in polynomial time when $k \leq 3$ [14], while it becomes PSPACE-complete for every fixed $k \geq 4$ [8]. Notably, this rule is closely related to Glauber dynamics in statistical physics, where a Markov chain is defined over the space of proper k-colorings of a graph G: at each step, a vertex is selected uniformly at random and recolored with a randomly chosen color such that the resulting coloring remains proper. See Sokal [33] for an introduction to the Potts model and its connections to graph coloring.

The second widely studied rule is *Kempe chain recoloring*. A new proper k-coloring is obtained by selecting a connected component C of the subgraph of G induced by two color classes (i.e., a *Kempe chain*) and swapping the two colors within C. Note that when C consists of a single vertex, this operation is equivalent to single-vertex recoloring. While k-Coloring Reconfiguration under this rule is solvable in polynomial time when $k \leq 2$ [27] (in fact, the answer is always "Yes"), it becomes PSPACE-complete for every fixed $k \geq 3$ [6]. Kempe chain recoloring was originally introduced by Kempe in 1879 in an attempt to prove the Four Color Theorem [24]. Although his proof was later found to be flawed, the technique has continued to play a central role in graph coloring theory [5, 27], statistical physics [28, 29], and the study of mixing times of Markov chains [36].

These results highlight a key aspect of reconfiguration problems: even when the definition of feasible solutions remains the same, the computational complexity of finding a reconfiguration sequence can vary greatly depending on the selected reconfiguration rule.

1.2 Our Contribution

In this paper, we introduce a new reconfiguration rule, called *color swapping* (CS) for Coloring Reconfiguration. A new proper k-coloring is obtained from a given one by swapping the colors of the endpoints of a single edge uv in G, so that the resulting coloring remains proper. For example, Figure 1 shows a reconfiguration sequence between f_s and f_t under CS, hence it is a yes-instance. We refer to Coloring Reconfiguration and k-Coloring Reconfiguration under CS as CRCS and k-CRCS, respectively. The color swapping rule can be seen as a restricted variant of Kempe chain recoloring, where each Kempe

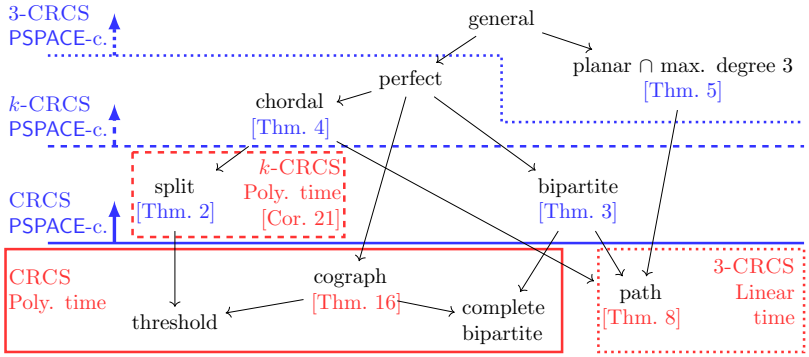


Figure 2 Our results for graph classes. Each arrow represents the inclusion relationship between classes: $A \to B$ means that the graph class B is a proper subclass of the graph class A.

chain is limited to exactly two vertices. Interestingly, this rule is also studied in statistical physics as *(local) Kawasaki dynamics*, which models dynamics in the fixed-magnetization Ising model [12, 23, 25].

The contribution of this paper is an analysis of the computational complexity of CRCS and k-CRCS; for an overview of our results, we refer to Figure 2. First, we prove that CRCS is PSPACE-complete even when the input graph is restricted to bipartite or split graphs. Furthermore, we show that there exists a positive integer k_0 such that for every fixed $k \geq k_0$, k-CRCS becomes PSPACE-complete even when the input graph is restricted to chordal graphs, which is a superclass of split graphs.

We also establish a complexity dichotomy with respect to the number k of colors: k-CRCS is PSPACE-complete for any fixed $k \geq 3$, whereas for $k \leq 2$, the problem can be solved in polynomial time. In particular, we show that 3-CRCS is PSPACE-complete even when restricted to planar graphs with maximum degree 3 and bounded bandwidth.

Complementing these hardness results, we also present several positive results. We first show that 3-CRCS can be solved in linear time on path graphs. To this end, we introduce an invariant for proper 3-colorings of paths, and design a linear-time algorithm that checks whether two input colorings have the identical invariants. While the algorithm is simple, its correctness requires a non-trivial argument.

Next, we show that CRCS can be solved in polynomial time on cographs. Our algorithm is based on a recursive procedure over the cotree of the input cograph, inspired by prior work [22] for Independent Set Reconfiguration on cographs. To adapt this approach to our problem, we introduce a new notion called *extended k-colorings* as a generalization of k-colorings.

Finally, we show that k-CRCS on split graphs is polynomial-time solvable for any fixed k. This contrasts with the PSPACE-hardness of CRCS on split graphs when k is unbounded.

Due to the space limitation, we omit the proofs of statements with the symbol $\star$ marks, which can be found in the full version of this paper.

1.3 Related Work

As mentioned in Section 1.1, the complexity of k-Coloring Reconfiguration has been extensively investigated with respect to various graph classes. Under both the single-vertex recoloring and Kempe chain recoloring rules, the problem is known to be PSPACE-complete in general. Moreover, stronger hardness results and polynomial-time algorithms have been established for specific graph classes.

Under the single-vertex recoloring rule, the problem remains PSPACE-complete even on bipartite planar graphs [8] and chordal graphs [17]. On the positive side, it is solvable in polynomial time for 2-degenerate graphs, q-trees (for fixed q), trivially perfect graphs, and split graphs [17]. Under the Kempe chain recoloring rule, the problem is PSPACE-complete even on planar graphs with maximum degree 6 [6]; however, it is polynomial-time solvable on chordal graphs, bipartite graphs, and cographs [6]. Several other algorithmic and structural aspects of k-Coloring Reconfiguration have also been studied, including finding a shortest reconfiguration sequence [9, 21] and bounding its length [1, 3, 10, 13].

The term *color swapping* is inspired by the Colored Token Swapping problem [7, 38], a reconfiguration problem involving token placements. In that problem, one is given a graph with an initial and a target coloring, which *need not be proper*, and the goal is to transform the initial coloring into the target one using the minimum number of swaps between tokens on adjacent vertices. Since feasibility is not restricted to proper colorings, this can be viewed as a variant of our reconfiguration problem in which the feasibility condition is relaxed. Yamanaka et al. [38] showed that Colored Token Swapping is NP-hard even when the number of colors is exactly three.

2 Preliminaries

For a positive integer k, we write $[k] = \{1, 2, \ldots, k\}$. For sets X and Y, the *symmetric difference* of X and Y is defined as $X \triangle Y \coloneqq (X \setminus Y) \cup (Y \setminus X)$. For a map $f\colon X \to Y$ and an element $y \in Y$, the *preimage* of y under f is defined as $f^{-1}(y) \coloneqq \{x \in X \mid f(x) = y\}$.

Let $G = (V, E)$ be an undirected graph. We use $V(G)$ and $E(G)$ to denote the vertex set and edge set of G, respectively. For a vertex v of G, $N_G(v)$ and $N_G[v]$ denote the *open neighborhood* and the *closed neighborhood* of v in G, respectively; that is, $N_G(v) = \{u \in V \mid uv \in E\}$ and $N_G[v] = N_G(v) \cup \{v\}$. For a vertex set $X \subseteq V$, we define $N_G(X) = \{v \in V \setminus X \mid u \in X, uv \in E(G)\}$ and $N_G[X] = N_G(X) \cup X$.

For a positive integer k, a *(proper) k-coloring* of a graph G is a map $f\colon V(G) \to [k]$ that assigns different colors to adjacent vertices; in other words, $f(u) \neq f(v)$ for every edge $uv \in E(G)$. An *independent set* of a graph G is a subset $I \subseteq V(G)$ such that for all $u, v \in I$, $uv \notin E(G)$ holds. A *clique* of a graph G is a subset $C \subseteq V(G)$ such that for all $u, v \in C$ with $u \neq v$, $uv \in E(G)$ holds.

2.1 Our Problems

For two proper colorings f and f' of a graph G, we say that f and f' are *adjacent* under *Color Swapping* (denoted by CS) if and only if the following two conditions hold:

1. There exists exactly one edge $uv \in E(G)$ such that $f(u) = f'(v)$ and $f(v) = f'(u)$, and
2. For all other vertices $w \in V(G) \setminus \{u, v\}$, we have $f(w) = f'(w)$.

Intuitively, f' can be obtained from f by swapping the colors of two adjacent vertices.

A sequence $f_0, f_1, \ldots, f_\ell$ of proper k-colorings of G with $f_0 = f$ and $f_\ell = f'$ is called a *reconfiguration sequence* between f and f' if f_{i-1} and f_i are adjacent under CS for all $i \in [\ell]$. We say that f and f' are *reconfigurable* if such a sequence exists. We now define the Coloring Reconfiguration under Color Swapping problem (CRCS for short). In CRCS, we are given a graph G, a positive integer k, and two proper k-colorings f_s and f_t of G. The goal is to determine whether there exists a reconfiguration sequence between f_s and f_t. For a fixed positive integer k, the k-CRCS problem is the special case of CRCS in which the two input colorings are k-colorings.

An instance (G, k, f_s, f_t) of CRCS is said to be *valid* if, for each color $i \in [k]$, the number of vertices assigned color i is the same in f_s and f_t; that is, $|f_s^{-1}(i)| = |f_t^{-1}(i)|$. Note that if f_s and f_t are reconfigurable, then (G, k, f_s, f_t) must be valid. This observation implies that if an instance (G, k, f_s, f_t) of CRCS is not valid, we can immediately return "NO." Therefore, throughout this paper, we assume that all instances of CRCS are valid.

It is easy to see that k-CRCS for $k \leq 2$ can be solved in polynomial time.

▶ **Observation 1** ($\star$). *k-CRCS can be solved in polynomial time for $k \leq 2$.*

3 PSPACE-completeness

We first observe that CRCS is solvable using polynomial space, i.e., CRCS belongs to the class PSPACE. This follows from the equivalence PSPACE = NPSPACE, which is a consequence of Savitch's theorem [32]. To see the membership of PSPACE, consider a reconfiguration sequence for CRCS as a certificate. Our polynomial-space algorithm reads each k-coloring in the sequence one by one and verifies that (i) each coloring is proper and (ii) each pair of consecutive colorings is adjacent under CS. Since each of these checks can be performed using polynomial space, this implies that CRCS can be solved in nondeterministic polynomial space. Therefore, by PSPACE = NPSPACE, we conclude that CRCS is in PSPACE.

In this section, we present polynomial-time reductions from three different problems to establish the PSPACE-completeness of our problem for several graph classes. Specifically, we reduce from the following problems: the Token Sliding problem in Section 3.1, the Coloring Reconfiguration problem in Section 3.2, and the Nondeterministic Constraint Logic problem in Section 3.3.

3.1 Reduction from Token Sliding

In this subsection, we present a polynomial-time reduction from the Token Sliding problem to CRCS. Token Sliding is also known as the Independent Set Reconfiguration under Token Sliding [18, 22].

Let G be a graph, and let $I \subseteq V(G)$ be a vertex subset of G. Recall that I is an *independent set* of G if no two vertices in I are adjacent; that is, $uv \notin E(G)$ for all $u, v \in I$. Two independent sets I and J of the same size are said to be *adjacent* under *token sliding* if and only if $|I \triangle J| = 2$ and the two vertices in $I \triangle J$ are joined by an edge in G.

In the Token Sliding problem, we are given a graph G and two independent sets $I_s, I_t \subseteq V(G)$ of the same size. The goal is to determine whether there exists a sequence $I_0, I_1, \ldots, I_\ell$ of independent sets of G such that $I_s = I_0$ and $I_t = I_\ell$, and for every $i \in [\ell]$, I_{i-1} and I_i are adjacent.

Token Sliding is known to be PSPACE-complete even when restricted to split graphs [2] or bipartite graphs [26]. A graph is *split* if its vertices can be partitioned into a clique and an independent set, and *bipartite* if its vertices can be partitioned into two independent sets.

We first show the following theorem for split graphs.

▶ **Theorem 2.** CRCS *is* PSPACE*-complete for split graphs.*

Proof. We have already observed that the problem is in PSPACE. To show the PSPACE-hardness, we give a polynomial-time reduction from Token Sliding on split graphs to CRCS.

Let (G, I_s, I_t) be an instance of Token Sliding such that $|I_s| = |I_t|$ and $G = (V, E)$ is a split graph with vertex partition (C, S), where C is a clique of G and S is an independent set of G. Assume that $|I_s| = |I_t| \geq 2$. Note that Token Sliding remains PSPACE-complete under this assumption since the problem is trivial when $|I_s| = |I_t| = 1$.

We construct an instance (G', k, f_s, f_t) of CRCS as follows. Let G' be the graph obtained from G by adding a new vertex u that is adjacent to all vertices in $V(G)$. That is, let $G' = (V', E')$ with $V' = V \cup \{u\}$ and $E' = E \cup \{uv \mid v \in V\}$. Since u is adjacent to all vertices in C, the set $C \cup \{u\}$ is a clique of G', and S is an independent set of G'. Thus, G' is also a split graph.

Let $k = |V'| - |I_s| + 1 = |V'| - |I_t| + 1$. We now define proper k-colorings f_s and f_t of G'. For each $v \in I_s$, set $f_s(v) = 1$. Then, assign a distinct color from $[k] \setminus \{1\}$ arbitrarily to each vertex in $V' \setminus I_s$ so that f_s is a proper k-coloring. Similarly, define f_t by setting $f_t(v) = 1$ for each $v \in I_t$, and assigning a distinct color from $[k] \setminus \{1\}$ arbitrarily to each vertex in $V' \setminus I_t$ so that f_t is also a proper k-coloring. Since $|V' \setminus I_s| = k - 1$ and $|V' \setminus I_t| = k - 1$, such proper k-colorings f_s and f_t exist.

This completes the construction of the instance (G', k, f_s, f_t). We claim that (G, I_s, I_t) is a yes-instance of Token Sliding if and only if (G', k, f_s, f_t) is a yes-instance of CRCS.

We first show the "only if" direction. Suppose that (G, I_s, I_t) is a yes-instance of Token Sliding. Then, there exists a sequence of independent sets $I_0, I_1, \ldots, I_\ell$ of G, with $I_0 = I_s$ and $I_\ell = I_t$, and for each $i \in [\ell]$, I_{i-1} and I_i are adjacent under token sliding. For each $i \in \{0, 1, \ldots, \ell\}$, we construct a proper k-coloring f_i of G' from I_i, following the same construction as for f_s and f_t: for each $v \in I_i$, set $f_i(v) = 1$, and assign a distinct color from $[k] \setminus \{1\}$ arbitrarily to each vertex in $V' \setminus I_i$ so that f_i is a proper k-coloring of G'.

We claim that for all $i \in [\ell]$, f_{i-1} and f_i are reconfigurable under CS. Let f' be the coloring obtained from f_{i-1} by exchanging the colors assigned to the vertices $v \in I_{i-1} \setminus I_i$ and $w \in I_i \setminus I_{i-1}$. Since each vertex in I_{i-1} is assigned the color 1 in f_{i-1}, and all other vertices are assigned distinct colors from $[k] \setminus \{1\}$, the modified coloring f' remains a proper k-coloring of G'. Moreover, since I_{i-1} and I_i are adjacent under token sliding, we have $vw \in E(G')$. Thus, f_{i-1} and f' are adjacent under CS.

We now show that f' and f_i are reconfigurable under CS. Recall that $f'(v) = f_i(v) = 1$ for all $v \in I_i$. Thus, f' and f_i may differ only on the vertices in $V(G') \setminus I_i$. Note that the restrictions of f' and f_i to $V(G') \setminus I_i$ are both bijective.

By construction, $G'[V(G') \setminus I_i]$ is connected, since it contains a universal vertex u adjacent to all others. It is known that for any connected n-vertex graph and two bijective n-colorings, there exists a reconfiguration sequence of length $O(n^2)$ between them under CS [37]. Applying this result, we can reconfigure f' into f_i using color swaps only within $V(G') \setminus I_i$. Thus, f_{i-1} and f_i are reconfigurable under CS. This implies that we can construct a reconfiguration sequence between f_s and f_t under CS. Therefore, (G', k, f_s, f_t) is a yes-instance of CRCS.

We now prove the "if" direction. Suppose that there exists a reconfiguration sequence between f_s and f_t under CS. Let $f_0, f_1, \ldots, f_\ell$ be the reconfiguration sequence, where $f_0 = f_s$ and $f_\ell = f_t$. For each $i \in \{0, 1, \ldots, \ell\}$, define $I_i = f_i^{-1}(1)$. Since the number of vertices colored 1 remains constant throughout the sequence, it follows that $|I_i| = |I_s| = |I_t| \geq 2$ for all i. By construction, the vertex u is adjacent to all other vertices in G', and thus cannot be assigned color 1 in any f_i; hence, $I_i \subseteq V(G)$. Moreover, since f_i is a proper k-coloring of G, the set I_i forms an independent set of G', and consequently also an independent set of G.

For each $i \in [\ell]$, since two consecutive proper k-colorings f_{i-1} and f_i are adjacent under CS, we can observe that I_{i-1} and I_i are either identical or adjacent under token sliding. By deleting any consecutive duplicate independent sets from the $I_0, I_1, \ldots, I_\ell$, we obtain the desired sequence between I_s and I_t. Therefore, (G, I_s, I_t) is a yes-instance of Token Sliding.

This completes the proof of Theorem 2. ◀

The similar result holds for bipartite graphs.

▶ **Theorem 3.** CRCS *is* PSPACE*-complete for bipartite graphs.*

Proof. We have already observed that the problem is in PSPACE. To show the PSPACE-hardness, we give a polynomial-time reduction from Token Sliding on bipartite graphs to CRCS.

Let (G, I_s, I_t) be an instance of Token Sliding, where G is a bipartite graph with a bipartition (S, T), and both S and T are independent sets of G. We construct an instance (G', k, f_s, f_t) of CRCS as follows (see also Figure 3).

First, we construct G' by adding three new vertices x_1, x_2, x_3 to S, and three new vertices y_1, y_2, y_3 to T. We then make x_1 adjacent to all vertices in $T \cup \{y_1, y_2, y_3\}$, and y_1 adjacent to all vertices in $S \cup \{x_1, x_2, x_3\}$. Since $S \cup \{x_1, x_2, x_3\}$ and $T \cup \{y_1, y_2, y_3\}$ are independent sets of G', the resulting graph G' is also bipartite.

We set $k = |V(G')| - |I_s| - 3$, and define proper k-colorings f_s and f_t of G' as follows. For the initial coloring f_s, assign color 1 to every vertex in $I_s \cup \{x_2, x_3, y_2, y_3\}$. Then, assign a distinct color from $[k] \setminus \{1\}$ arbitrarily to each vertex in $V(G') \setminus (I_s \cup \{x_2, x_3, y_2, y_3\})$ so that f_s is a proper k-coloring of G'. Similarly, for the target coloring f_t, assign color 1 to every vertex in $I_t \cup \{x_2, x_3, y_2, y_3\}$. Then, assign a distinct color from $[k] \setminus \{1\}$ arbitrarily to each vertex in $V(G') \setminus (I_t \cup \{x_2, x_3, y_2, y_3\})$ such that f_t is also a proper k-coloring of G'. Note that both $V(G') \setminus (I_s \cup \{x_2, x_3, y_2, y_3\})$ and $V(G') \setminus (I_t \cup \{x_2, x_3, y_2, y_3\})$ contain exactly $(k-1)$ vertices. Moreover, both $(I_s \cup \{x_2, x_3, y_2, y_3\})$ and $(I_t \cup \{x_2, x_3, y_2, y_3\})$ form independent sets of G'. Thus, it is always possible to assign the remaining $(k-1)$ colors so that both f_s and f_t are proper k-colorings of G'.

This completes the construction of the instance (G', k, f_s, f_t). We then claim that (G, I_s, I_t) is a yes-instance of Token Sliding if and only if (G', k, f_s, f_t) is a yes-instance of CRCS.

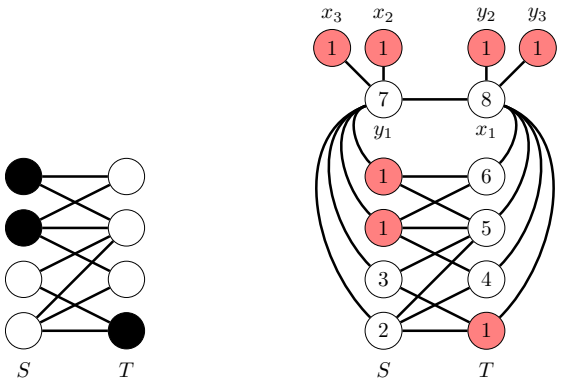
We first show the "only if" direction. Suppose that (G, I_s, I_t) is a yes-instance of Token Sliding. Then, there exists a sequence of independent sets $I_0, I_1, \ldots, I_\ell$ of G, with $I_0 = I_s$ and $I_\ell = I_t$, and for each $i \in [\ell]$, I_{i-1} and I_i are adjacent under token sliding. For each $i \in \{0, 1, \ldots, \ell\}$, we construct a proper k-coloring f_i of G' from I_i, following the same construction as for f_s and f_t. for the coloring f_i, assign color 1 to every vertex in $I_i \cup \{x_2, x_3, y_2, y_3\}$. Then, assign a distinct color from $[k] \setminus \{1\}$ arbitrarily to each vertex in $V(G') \setminus (I_i \cup \{x_2, x_3, y_2, y_3\})$ such that f_i is also a proper k-coloring.

We claim that f_{i-1} and f_i are reconfigurable under CS. Let f' be the coloring obtained from f_{i-1} by exchanging the colors assigned to the vertices $v \in I_{i-1} \setminus I_i$ and $w \in I_i \setminus I_{i-1}$. Since each vertex in $I_i \cup \{x_2, x_3, y_2, y_3\}$ is assigned the color 1, and all other vertices are assigned distinct colors from $[k] \setminus \{1\}$ by f', the modified coloring f' remains a proper k-coloring of G'. Moreover, since I_{i-1} and I_i are adjacent under token sliding, we have $vw \in E(G')$. Hence, f_i and f' are adjacent under CS.

We now prove that f' and f_i are reconfigurable under CS. Recall that $f'(v) = f_i(v) = 1$ for all $v \in I_i \cup \{x_2, x_3, y_2, y_3\}$ Thus, f' and f_i may differ only on the vertices in $[k] \setminus \{1\}$. Note that the restrictions of f' and f_i to $V(G') \setminus I_i$ are both bijective.

Recall that x_1 is adjacent to every vertex in $T \cup \{y_1, y_2, y_3\}$ and y_1 is adjacent to every vertex in $S \cup \{x_1, x_2, x_3\}$, so the subgraph induced by $V(G') \setminus (I_i \cup \{x_2, x_3, y_2, y_3\})$ is connected. As shown in the proof of Theorem 2 and using the result from [37], we see that f' and f_i are reconfigurable under CS, and thus so are f_{i-1} and f_i. By repeating this process for all $i \in [\ell]$, we obtain a reconfiguration sequence from f_s to f_t under CS. Therefore, (G', k, f_s, f_t) is a yes-instance of CRCS.

We now prove the "if" direction. Suppose that there exists a reconfiguration sequence between f_s and f_t under CS. Let $f_0, f_1, \ldots, f_\ell$ be the reconfiguration sequence, where $f_0 = f_s$ and $f_\ell = f_t$.



(a) (b)

Figure 3 (a) The bipartite graph G for an instance of the Token Sliding problem, along with its initial independent set I, where the vertices in I are marked in black. The vertex set of the graph G is partitioned into independent sets S and T. (b) The bipartite graph G', constructed from G, has a bipartition $V(G') = (S \cup \{x_1, x_2, x_3\}) \cup (T \cup \{y_1, y_2, y_3\})$, where both parts are independent sets. Each vertex is labeled with its color in the initial proper k-coloring f_s derived from I.

Since x_2 and x_3 share the same neighbor y_1 and are both assigned color 1 in f_s, no color swap involving x_2 or x_3 is allowed throughout the sequence; that is, $f_i(x_2) = f_i(x_3) = 1$ for all $i \in \{0, 1, \ldots, \ell\}$. By symmetry, we also have $f_i(y_2) = f_i(y_3) = 1$, which implies that x_1 and y_1 are never assigned color 1. For each $i \in \{0, 1, \ldots, \ell\}$, define $I_i = f_i^{-1}(1)$ and $I'_i = I_i \setminus \{x_2, x_3, y_2, y_3\}$. Then, $|I'_i| = |I_s| = |I_t|$, and since I_i is an independent set in G', it follows that I'_i is also an independent set in G.

Furthermore, since f_{i-1} and f_i are adjacent under CS for each $i \in [\ell]$, the sets I'_{i-1} and I'_i are either adjacent under token sliding or identical. By removing consecutive duplicates from the sequence $I'_0, I_1, \ldots, I'_\ell$, we obtain a reconfiguration sequence of independent sets from I_s to I_t under token sliding. Therefore, (G, I_s, I_t) is a yes-instance of Token Sliding.

This completes the proof of Theorem 2. ◀

3.2 Reduction from Coloring Reconfiguration

In this subsection, we give a polynomial-time reduction from the k-Coloring Reconfiguration problem under single-vertex recoloring to k-CRCS.

Let k be a positive integer. In k-Coloring Reconfiguration under single-vertex recoloring, we are given a graph G and two k-colorings g and g' of G. The goal is to determine whether there exists a sequence of k-colorings $g_0, g_1, \ldots, g_\ell$ with $g_0 = g$ and $g_\ell = g'$ such that for each $i \in [\ell]$, the colorings g_{i-1} and g_i differ at exactly one vertex; that is, $|\{v \in V(G) \mid g_{i-1}(v) \neq g_i(v)\}| = 1$. We simply call the problem k-Coloring Reconfiguration. It is known that there exists a positive integer k_0 such that, for any fixed $k \geq k_0$, k-Coloring Reconfiguration is PSPACE-complete on chordal graphs [17]. Recall that a graph is *chordal* if it contains no induced cycle of length at least 4.

We claim the following theorem.

▶ **Theorem 4.** *There exists a positive integer k_0 such that, for any fixed $k \geq k_0$, k-CRCS is* PSPACE*-complete on chordal graphs.*

Proof. We have already observed that the problem is in PSPACE. To show the PSPACE-hardness, we give a polynomial-time reduction from k-Coloring Reconfiguration on chordal graphs to k-CRCS.

Let (G, g_s, g_t) be an instance of k-Coloring Reconfiguration, where G is a chordal graph. We construct an instance (G', f_s, f_t) of k-CRCS as follows (see also Figure 4). For each vertex $v \in V(G)$, we add $(k-1)$ new vertices $v_1, v_2, \ldots, v_{k-1}$ and make $\{v, v_1, v_2, \ldots, v_{k-1}\}$ a clique in G'. Let $C_v = \{v, v_1, v_2, \ldots, v_{k-1}\}$. Note that the resulting graph G' is also chordal. We then define the k-coloring f_s as follows: for each $v \in V(G)$, set $f_s(v) = g_s(v)$, and for each vertex in $C_v \setminus \{v\}$, assign an arbitrary color distinct from $f_s(v)$ so that f_s is a proper k-coloring of G (which is always possible since $|C_v| = k$). Similarly, define $f_t(v) = g_t(v)$ for each $v \in V(G)$, and for each vertex in $C_v \setminus \{v\}$, assign an arbitrary color distinct from $g_t(v)$ so that f_t is a proper k-coloring of G.

This completes the construction of the instance (G', f_s, f_t). We then claim that (G, g_s, g_t) is a yes-instance of k-Coloring Reconfiguration if and only if (G', f_s, f_t) is a yes-instance of k-CRCS.

We first prove the "only if" direction. Suppose that (G, g_s, g_t) is a yes-instance of k-Coloring Reconfiguration. Then there exists a reconfiguration sequence $g_0, g_1, \ldots, g_\ell$ of proper k-colorings of G such that $g_0 = g_s$, $g_\ell = g_t$, and for each $i \in [\ell]$, g_{i-1} and g_i differ at exactly one vertex. For each $i \in \{0, 1, \ldots, \ell\}$, we construct a proper k-coloring f_i of G' from g_i, following the same construction as for f_s and f_t: for every $v \in V(G)$, set $f_i(v) = g_i(v)$, and for each $u \in C_v \setminus \{v\}$, assign an arbitrary color distinct from $f_i(v)$ so that f_i is a proper k-coloring of G'.

We claim that for all $i \in [\ell]$, f_{i-1} and f_i are reconfigurable under CS. Indeed, let $u \in V(G)$ be the unique vertex such that $g_{i-1}(u) \neq g_i(u)$. By construction, we have $f_{i-1}(u) \neq f_i(u)$, while $f_{i-1}(v) = f_i(v)$ for all $v \in V(G) \setminus \{u\}$. Let $w \in C_u$ be the unique vertex such that $f_{i-1}(w) = f_i(u)$. We construct a k-coloring f' from f_{i-1} by swapping the colors of u and w. Note that $f'(v) = f_i(v)$ for all $v \in V(G) \setminus \{u\}$.

Recall that for each $v \in V(G)$, the clique C_v in G' contains exactly k vertices, and in both colorings f_i and f', the vertices of C_v receive pairwise distinct colors. Since vertices in $C_v \setminus \{v\}$ are adjacent only to those in C_v, we can freely perform color swaps within C_v without affecting outside C_v. Thus, we can reconfigure f' into f_i by performing a sequence of color swaps only within cliques C_v for $v \in V(G)$. This implies that f_{i-1} and f_i are reconfigurable under CS. Applying this argument for each $i \in [\ell]$ yields a reconfiguration sequence from f_s to f_t in G' under CS. Therefore, (G', f_s, f_t) is a yes-instance of k-CRCS.

We now prove the "if" direction. Suppose that there exists a reconfiguration sequence $f_0, f_1, \ldots, f_\ell$ between f_s and f_t, where $f_0 = f_s$ and $f_\ell = f_t$. For each $i \in \{0, 1, \ldots, \ell\}$, define the coloring g_i of G by setting $g_i(v) = f_i(v)$ for every $v \in V(G)$. Since f_i is a proper k-coloring of G', the construction guarantees that g_i is a proper k-coloring of G.

Observe that each C_w ($w \in V(G)$) is a clique of size k. Thus, any color swap occurs only on two vertices within some clique C_w. It follows that for every $i \in [\ell]$, the colorings g_{i-1} and g_i are either identical or differ at exactly one vertex of G. By removing any consecutive duplicate colorings, we obtain a desired sequence of proper k-colorings of G from g_s to g_t. Therefore, (G, g_s, g_t) is a yes-instance of k-Coloring Reconfiguration.

This completes the proof of Theorem 4. ◀

3.3 Reduction from Nondeterministic Constraint Logic

In this subsection, we prove Theorem 5 by a polynomial-time reduction from the Nondeterministic Constraint Logic problem [19, 35].

▶ **Theorem 5.** *For every fixed integer $k \geq 3$, k-CRCS is* PSPACE*-complete for planar graphs of maximum degree 3 and bounded bandwidth.*

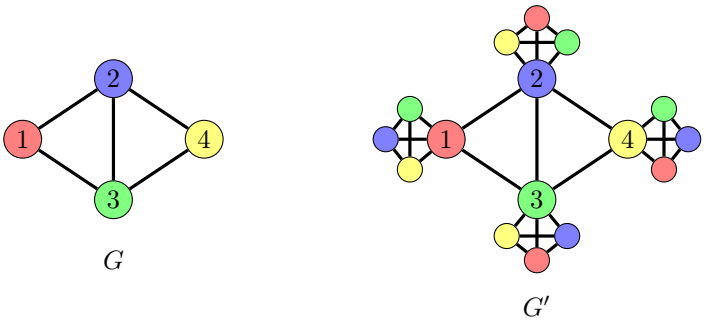


Figure 4 Construction of G from G' using four colors. Vertices of G are assigned a proper 4-coloring g_t, and those of G' the corresponding proper 4-coloring f_t.

We begin by formally defining the Nondeterministic Constraint Logic problem in Section 3.3.1. Next, we introduce an auxiliary gadget used in our reduction, which is described in Section 3.3.2. We then present the full reduction in Section 3.3.3, including the design of two types of gadgets: AND gadgets, and OR gadgets. Finally, we prove the correctness of the reduction in Section 3.3.4.

Note that our reduction is presented for $k = 3$; however, it holds analogously for cases where $k \geq 4$.

3.3.1 Definition of Nondeterministic Constraint Logic

A *Nondeterministic Constraint Logic (NCL) machine* is an undirected graph in which each edge is assigned a weight from $\{1, 2\}$ (see Figure 5 (a)). A *configuration* of an NCL machine is an orientation of its edges such that, at every vertex, the total weight of incoming edges is at least 2. Two configurations are said to be *adjacent* if they differ in the orientation of exactly one edge. In the Nondeterministic Constraint Logic problem, we are given an NCL machine M and two of its configurations, C_s and C_t. The goal is to determine whether there exists a sequence of configurations starting from C_s and ending at C_t, such that each consecutive pair of configurations in the sequence differs in the orientation of exactly one edge; that is, they are adjacent.

An NCL machine M is called an *AND/OR constraint graph* if it contains only two types of vertices: *AND vertices* and *OR vertices* (see again Figure 5 (a)). A vertex of degree 3 in M is an *AND vertex* if its incident three edges have weights 1, 1, and 2; see Figure 5 (b). Similarly, a vertex of degree 3 in M is defined as an *OR vertex* if all its incident three edges have weight 2; see Figure 5 (c). For the remainder of this paper, we will use the term *NCL machine* to refer specifically to an AND/OR constraint graph. It is known that the Nondeterministic Constraint Logic problem remains PSPACE-complete when the input NCL machine (AND/OR constraint graph) is restricted to be planar, of maximum degree 3, and of bounded bandwidth [35]. Recall that, for a graph G, the *bandwidth* of G is the minimum integer b such that G has a bijection $\pi\colon V(G) \to [|V(G)|]$ with $\max_{uv \in E(G)} |\pi(u) - \pi(v)| \leq b$.

3.3.2 Auxiliary Gadgets

Before presenting our full construction, we introduce auxiliary gadgets, called *forbidden pendants*, which prevent a vertex from being assigned a specific color.

Let $C = \{1, 2, 3\}$ be a set of colors. Consider a vertex x adjacent to a vertex y, where y is part of a 4-cycle, as illustrated in Figure 6. We consider two possible colorings of the 4-cycle in clockwise order starting from y: $(1, 2, 1, 3)$ or $(3, 1, 3, 2)$. Note that in each of these colorings, y is assigned color 1 or 3, respectively.

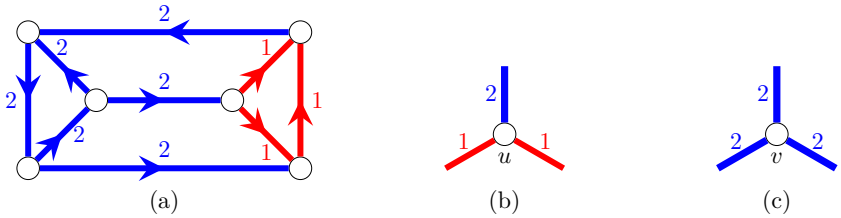


Figure 5 (a) A configuration of an NCL machine, (b) an AND vertex, and (c) an OR vertex. Edges of weight 2 are shown in blue lines, and edges of weight 1 in red lines. The NCL machine in (a) is an AND/OR constraint graph.

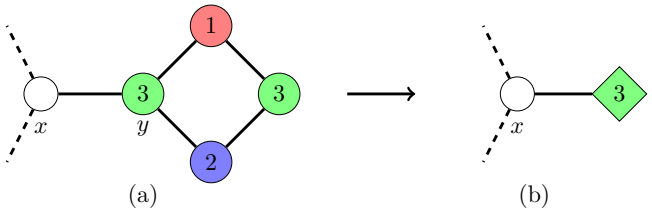


Figure 6 (a) An illustration of a 3-forbidden pendant for a vertex x, which ensures that x is never assigned color 3. (b) A simplified depiction of the gadget used to represent this pendant.

Since x is adjacent to y, it must be assigned a color different from that of y. In particular, x can be assigned only colors from the sets $\{2,3\}$ or $\{1,2\}$, respectively, depending on the color of y. Furthermore, observe that no valid color swaps can occur within the cycle or between x and y in any reconfiguration sequence. This ensures that the color of y remains fixed, and thus x is prevented from ever taking the same color as y.

If y is assigned color $c \in \{1,3\}$, we refer to this gadget as a *c-forbidden pendant* for x. For simplicity, we use the diagram shown in Figure 6 (b) to represent such a gadget.

3.3.3 AND/OR Gadgets and Our Reduction

Let $I = (M, C_s, C_t)$ be an instance of Nondeterministic Constraint Logic. Our reduction constructs a graph G by using two types of *vertex gadgets*, which simulate AND and OR vertices of M, respectively. Each vertex of M is replaced by the corresponding gadget according to its type.

For each edge $e = uv$ in M, the vertex gadgets for u and v each contain a special vertex, called a *port vertex*, which serves as an interface to the corresponding edge. These two port vertices, denoted by u_e and v_e, are connected by an edge referred to as a *port edge* (see Figure 7). Accordingly, the gadget corresponding to a vertex u of M with incident edges e_1, e_2, and e_3 includes three port vertices: u_{e_1}, u_{e_2}, and u_{e_3}.

Let u be an AND vertex in M, incident to one weight-2 edge e_1 and two weight-1 edges e_2 and e_3. We construct the corresponding AND gadget G_u, which consists of two internal vertices: u_0 and u_2^1, along with three port vertices: $u_1^1 = u_{e_1}$, $u_1^2 = u_{e_2}$, and $u_1^3 = u_{e_3}$. Each port vertex of G_u is adjacent to a 3-forbidden pendant (see Figure 8 (a)); hence, it can only be colored with color 1 or 2 in any proper coloring. Consequently, any color swap involving a port vertex in G_u must occur along the corresponding port edge.

Let u be an OR vertex in M, incident to three weight-2 edges e_1, e_2, and e_3. The corresponding OR gadget G_v consists of 12 vertices, denoted by v_j^i for $i \in [3]$ and $j \in [4]$. For each $i \in [3]$, the vertex v_4^i is a port vertex of G_v, that is, $v_4^i = v_{e_i}$. Moreover, every v_4^i is adjacent to a 3-forbidden pendant, while each of v_2^i and v_3^i is adjacent to a 1-forbidden pendant (see Figure 8(b)).

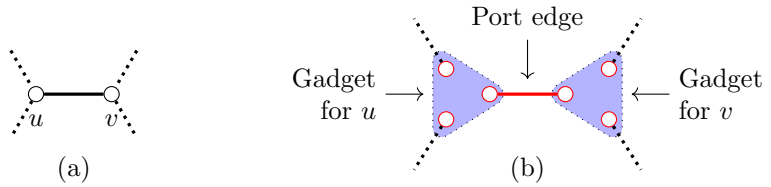


Figure 7 (a) An edge uv in G, and (b) its corresponding gadgets, where the port vertices are depicted by red circles and the shared port edge is depicted by a red line.

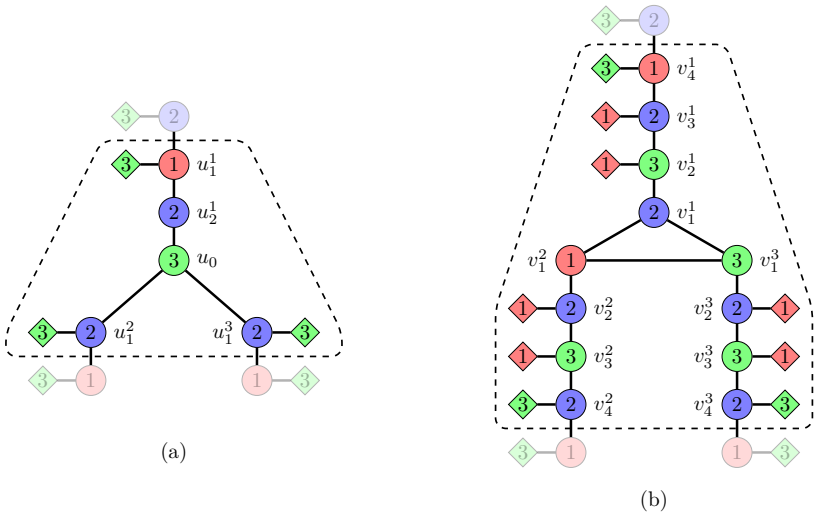


Figure 8 (a) The AND gadget and (b) the OR gadget, corresponding to (b) and (c) in Figure 5, respectively.

Similar to the AND gadget, each port vertex in G_v is restricted to colors 1 or 2 due to its adjacency to a 3-forbidden pendant. Consequently, any color swap involving a port vertex in G_v must occur along the corresponding port edge. Moreover, since both v_2^i and v_3^i for $i \in [3]$ are adjacent to 1-forbidden pendants, they must be assigned distinct colors: one must be colored 2 and the other 3. Thus, any color swap involving v_2^i or v_3^i can only occur on the edge $v_2^i v_3^i$.

Reduction

Let $I = (M, C_s, C_t)$ be an instance of Nondeterministic Constraint Logic. We construct a corresponding instance (G, f_s, f_t) of k-CRCS.

We begin by constructing a graph G from the NCL instance M as follows. For each AND or OR vertex in M, we replace it with the corresponding gadget as defined in Section 3.3.3. Then, for each edge $e = uv$ in M, we add a port edge between the corresponding port vertices u_e and v_e in the gadgets for u and v, respectively; see Figure 7.

Let G denote the resulting graph. Then, we observe the following.

▶ **Observation 6** ($\star$). *The constructed graph G is planar, of maximum degree* 3*, and has bounded bandwidth.*

We define two proper 3-colorings, f_s and f_t, of the constructed graph G. We begin by assigning colors to all c-forbidden pendants for $c \in \{1, 3\}$ according to the colorings described in Section 3.3.2.

Let e be an edge of M with an endpoint u. For each corresponding port vertex u_e, we set $f_s(u_e) = 1$ (resp. $f_t(u_e) = 1$) if the edge e is directed toward u in the configuration C_s (resp. C_t); otherwise, we assign $f_s(u_e) = 2$ (resp. $f_t(u_e) = 2$).

For each AND gadget G_u corresponding to an AND vertex u of M, we assign colors to the internal vertices u_2^1 and u_0 of G_u, which are depicted in Figure 8, so that f_s becomes a proper 3-coloring. That is, we set either $f_s(u_2^1) = 2$ and $f_s(u_0) = 3$, or $f_s(u_2^1) = 3$ and $f_s(u_0) = 2$. We define f_t similarly to f_s.

For each OR gadget G_v corresponding to an OR vertex v of M, we assign colors to the internal vertices of G_v depending on the coloring of its port vertices under f_s (resp. f_t). Specifically, for a given coloring of the port vertices, the internal vertices are colored according to one of the configurations illustrated in Figure 9, so that f_s (resp. f_t) becomes a proper 3-coloring. Since multiple valid internal colorings may exist for the same coloring of the port vertices of G_v, we may choose any such coloring arbitrarily.

This completes our polynomial-time reduction.

3.3.4 Correctness

Before proceeding to our proof, we provide an overview of the main ideas behind our reduction and outline the argument for its correctness.

Our reduction establishes a correspondence between the orientations of edges in a given instance of Nondeterministic Constraint Logic and the colorings in the constructed graph G. For each edge $e = uv$, we have associated two port vertices u_e and v_e in the gadgets for u and v, respectively. We interpret the edge as being oriented toward vertex v if u_e is assigned color 2, and as oriented toward vertex u if v_e is assigned color 2. Note that the coloring of each pair $\{u_e, v_e\}$ must be such that exactly one vertex is colored with 2 and the other with 1.

We briefly describe the behavior of the AND and OR gadgets. The AND gadget ensures that the port vertex u_1^1 can be colored with 2 only if both u_1^2 and u_1^3 are colored with 1. Once this condition is satisfied, the vertex u_2^1 can be recolored with 3 via a color swap with u_0.

Next, we explain the behavior of the OR gadgets. In an OR vertex of M, it suffices that at least one of the three incident edges is oriented inward. Accordingly, our gadget must only prohibit the configuration in which all three port vertices v_4^1, v_4^2, v_4^3 are simultaneously colored with 2. As shown in Figure 8(b), our construction enforces this constraint: if all three port vertices are colored with 2, then all intermediate vertices v_2^1, v_2^2, v_2^3 must also be colored with 2, which is impossible because the three vertices v_1^1, v_1^2, v_1^3 form a clique. Thus, at least one of the port vertices must be assigned color 1. See all proper colorings of the OR gadget shown in Figure 9.

To establish the correctness of our reduction, we present the following lemma.

▶ **Lemma 7** (⋆). *The instance (M, C_s, C_t) of* Nondeterministic Constraint Logic *is a yes-instance if and only if the constructed instance (G, f_s, f_t) of* 3-CRCS *is a yes-instance.*

4 Polynomial-time Algorithms

In this section, we present polynomial-time algorithms for CRCS and k-CRCS on paths, cographs, and split graphs.

4.1 Paths

We begin by presenting the following result for path graphs:

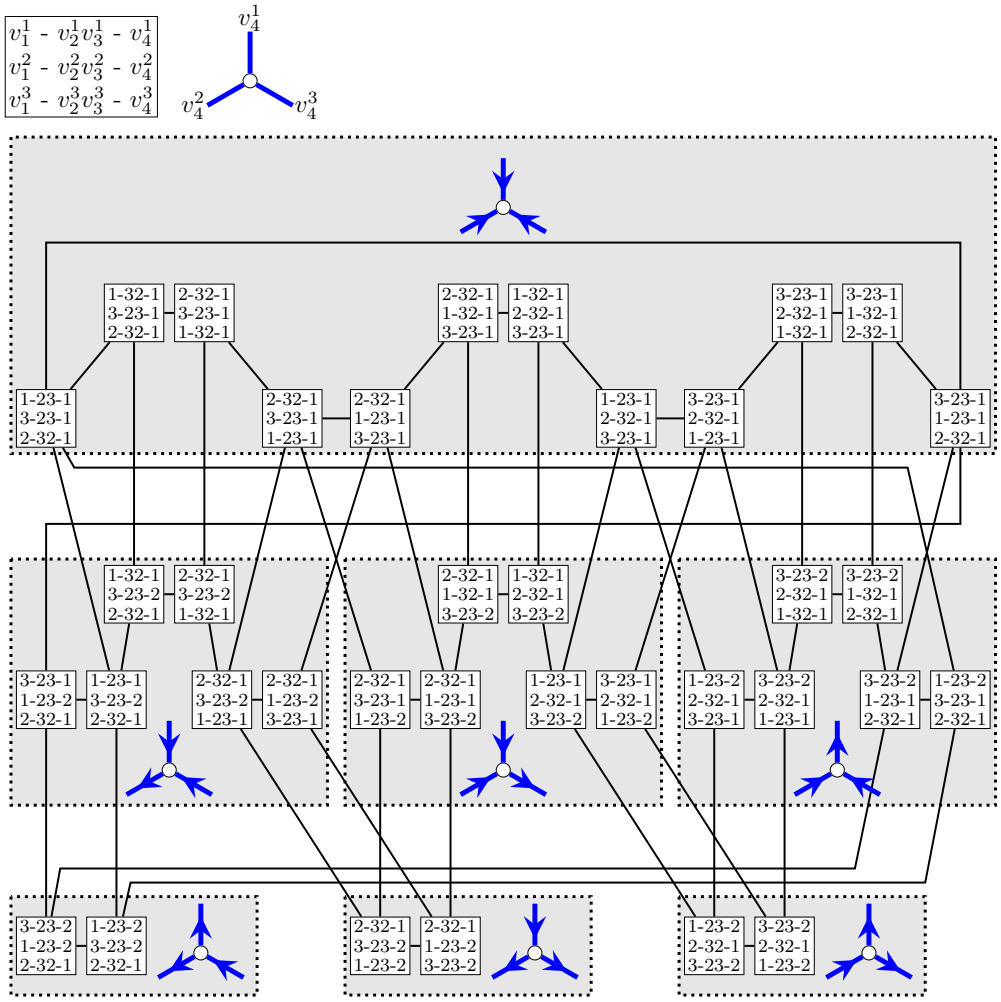


Figure 9 All valid orientations of the three edges incident to an OR vertex v, and the corresponding colorings of the OR gadget with their adjacency. Port vertices v_4^1, v_4^2, and v_4^3 are shown, and only the colors excluding those forbidden by c-forbidden pendants are indicated. In each row, no color swap is allowed between vertices separated by a hyphen due to the c-forbidden pendants.

▶ **Theorem 8.** 3-CRCS *can be solved in linear time for path graphs.*

To prove Theorem 8, we design a linear-time algorithm that solves 3-CRCS on path graphs.

In our algorithm, we compute and compare the invariants of the two input 3-colorings. If the invariants are identical, the algorithm returns `YES`; otherwise, it returns `NO`. Although the implementation of our algorithm is relatively simple, we emphasize that the core idea behind the algorithm is conceptually nontrivial, as it captures the essential structure preserved under CS.

4.1.1 Invariant for Coloring Strings

Before introducing the invariant, we first define several terms used throughout this subsection.

Given a string S, $S[i]$ denotes the i-th character of S, and $S[i, j]$ denotes the substring from the i-th to the j-th character (inclusive). If $i > j$, we define $S[i, j] := \mathsf{NIL}$, where NIL denotes the empty string (a string of length 0).

Let $P = v_1, v_2, \ldots, v_n$ be a path on n vertices. A string $S = S[1]S[2]\cdots S[n]$ is called a *coloring string* if there exists a proper 3-coloring f of P such that $S[i] = f(v_i)$ for all $i \in [n]$. We sometimes refer to S as the coloring string corresponding to f. Note that a coloring f can be encoded into its corresponding coloring string in $O(n)$ time.

We say that two consecutive characters in S are *swappable* if they can be exchanged to produce another coloring string S'. In this case, we say that S and S' are *adjacent*. If no such pair of swappable characters exists in S, then we say that S is *rigid*. Note that each swap of two characters precisely corresponds to a single color swapping operation. Finally, two coloring strings S and S' are said to be *reconfigurable* if there exists a sequence $S_0, S_1, \ldots, S_\ell$ of coloring strings such that $S_0 = S$, $S_\ell = S'$, and each consecutive pair S_{i-1}, S_i is adjacent for all $i \in [\ell]$.

We next observe a condition that determines whether two adjacent characters in a coloring string are swappable. Recall that a coloring string represents a proper 3-coloring of an n-vertex path.

▶ **Observation 9.** *Let $S = s_1 s_2 \cdots s_n$ be a coloring string. We can swap s_i and s_{i+1} in S for $i \in [n-1]$ if and only if the following conditions are satisfied:*

- *If $i = 1$, then the three characters s_1, s_2, s_3 are pairwise distinct.*
- *If $2 \le i \le n-2$, then $s_{i-1} = s_{i+2}$ holds.*
- *If $i = n-1$, then the three characters s_{n-2}, s_{n-1}, s_n are pairwise distinct.*

We now define the notion of *contraction*, which will be used to define our invariant for a coloring string. Let $S = s_1 s_2 \cdots s_n$ be a coloring string of length $n \ge 3$. We define the following three contraction operations:

(C1.) If $S = S[1, i-2]\, s_{i-1} s_i s_{i+1} s_{i+2}\, S[i+3, n]$ and $s_{i-1} = s_{i+2}$ for some $2 \le i \le n-2$, then S can be contracted to $S[1, i-2]\, s_{i+2}\, S[i+3, n]$.

(C2.) If s_1, s_2, s_3 are pairwise distinct, then $S = s_1 s_2 s_3\, S[4, n]$ can be contracted to $S[4, n]$.

(C3.) If s_{n-2}, s_{n-1}, s_n are pairwise distinct, then $S = S[1, n-3]\, s_{n-2} s_{n-1} s_n$ can be contracted to $S[1, n-3]$.

Note that in each of these cases, the contracted substring consists of three characters that are pairwise distinct.

Let $\mathsf{cont}(S)$ denote the set of all coloring strings that can be obtained from S by applying a single contraction operation. Each contraction reduces the length of the string by exactly 3, i.e., for every $S' \in \mathsf{cont}(S)$, we have $|S'| = |S| - 3$. We now define the invariant $\mathsf{inv}(S)$ of a coloring string S recursively as follows:

$$\mathsf{inv}(S) = \begin{cases} \mathsf{NIL} & \text{if } |S| \le 2, \\ \mathsf{inv}(S') & \text{if there exists } S' \in \mathsf{cont}(S), \\ S & \text{otherwise. (i.e., } S \text{ is rigid)} \end{cases}$$

The following lemma shows that $\mathsf{inv}(S)$ is uniquely determined regardless of the order in which the contractions are applied.

▶ **Lemma 10 (⋆).** *For any coloring string S, the invariant $\mathsf{inv}(S)$ is well-defined.*

A straightforward implementation of computing the invariant of a coloring string would take $O(n^2)$ time, as each contraction step may require scanning the entire string, and up to $O(n)$ such steps may be needed. However, the procedure can be implemented in linear time using a stack to efficiently simulate the recursive contractions.

▶ **Lemma 11 (⋆).** *For any coloring string S of length n, the invariant $\mathsf{inv}(S)$ can be computed in $O(n)$ time.*

4.1.2 Correctness of Algorithm

In the following, we discuss the correctness of our algorithm presented in Section 4.1.1. We begin with the following lemma, which states that taking a single swap of characters preserves the invariant.

▶ **Lemma 12** (⋆). *Let S and S' be two adjacent coloring strings. Then,* $\mathsf{inv}(S) = \mathsf{inv}(S')$.

The next two lemmas are crucial for our converse direction: if two coloring strings have the identical invariant, then they are reconfigurable.

▶ **Lemma 13** (⋆). *Let S be a coloring string of length at least 3. If S is not rigid, then there exists a coloring string S' such that $S'[1], S'[2], S'[3]$ are pairwise distinct, and S and S' are reconfigurable.*

▶ **Lemma 14** (⋆). *Let S and S' be coloring strings, and let w_1, w_2, w_3 and w'_1, w'_2, w'_3 be two triples of pairwise distinct characters such that $w_1w_2w_3S$ and $w'_1w'_2w'_3S'$ are coloring strings. If S and S' are reconfigurable, then $w_1w_2w_3S$ and $w'_1w'_2w'_3S'$ are also reconfigurable.*

The following is the main theorem in this subsection.

▶ **Theorem 15.** *Let (P, f_s, f_t) be a valid instance of* 3-CRCS *such that P is a path of n-vertices, and let S and S' be the coloring strings corresponding to f_s and f_t, respectively. Then,* $\mathsf{inv}(S) = \mathsf{inv}(S')$ *if and only if S and S' are reconfigurable.*

Proof. The "if" direction follows directly from Lemma 12. Hence, we now prove the "only-if" direction. We show that S and S' are reconfigurable if $\mathsf{inv}(S) = \mathsf{inv}(S')$, by induction on the length of S (and S').

For the base case where $|S| = |S'| \in \{0, 1, 2\}$, we always have $\mathsf{inv}(S) = \mathsf{inv}(S') = \mathsf{NIL}$, and it is clear that S and S' are reconfigurable.

For the induction step, assume that the claim holds for all coloring strings shorter than n. Now consider $n = |S| = |S'| \geq 3$. Since $\mathsf{inv}(S) = \mathsf{inv}(S')$, either both S and S' are rigid, or both are non-rigid. If both are rigid, then $S = \mathsf{inv}(S) = \mathsf{inv}(S') = S'$, so S and S' are identical and thus trivially reconfigurable.

Suppose S and S' are not rigid. By Lemma 13, there exist coloring strings S_x and S_y such that the first three characters of each string are pairwise distinct, and S is reconfigurable to S_x, while S' is reconfigurable to S_y. Note that S_x and S_y can be contracted to $S_x[4, n]$ and $S_y[4, n]$, respectively. Since $\mathsf{inv}(S_x[4, n]) = \mathsf{inv}(S_x) = \mathsf{inv}(S) = \mathsf{inv}(S') = \mathsf{inv}(S_y) = \mathsf{inv}(S_y[4, n])$, by the induction hypothesis, $S_x[4, n]$ and $S_y[4, n]$ are reconfigurable.

Applying Lemma 14, it follows that S_x and S_y are also reconfigurable. Consequently, by the transitivity of reconfigurability, S and S' are reconfigurable. ◀

4.2 Cographs

We begin by defining the class of *cographs*, also known as P_4-free graphs [15]. For two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, their *disjoint union* is defined as $G_1 \oplus G_2 = (V_1 \cup V_2, E_1 \cup E_2)$, and their *join* is defined as $G_1 \otimes G_2 = (V_1 \cup V_2, E_1 \cup E_2 \cup \{v_1v_2 \mid v_1 \in V_1, v_2 \in V_2\})$. A graph $G = (V, E)$ is a *cograph* if it can be constructed recursively according to the following rules:

1. A graph consisting of a single vertex is a cograph.
2. If G_1 and G_2 are cographs, then their disjoint union $G_1 \oplus G_2$ is also a cograph.
3. If G_1 and G_2 are cographs, then their join $G_1 \otimes G_2$ is also a cograph.

This inductive definition yields a canonical tree representation called a *cotree*, where each leaf corresponds to a vertex of the graph, and each internal node is labeled as either a disjoint union node ($\oplus$) or a join node ($\otimes$). Note that, by the definition of cographs, a cotree is a binary tree. For a cograph G, let T_G denote a cotree corresponding to G, and for a node $t \in V(T_G)$, let G_t denote the subgraph of G induced by the leaves of the subtree of T_G rooted at t. It is known that a cotree of a given cograph can be computed in linear time [34].

In this subsection, we provide a polynomial-time algorithm for CRCS on cographs.

▶ **Theorem 16.** CRCS *can be solved in polynomial time for cographs.*

4.2.1 Extended k-Colorings

To describe our algorithm, we introduce the notion of *extended colorings*. Let k be a positive integer. An *extended k-coloring* of a graph G is a map $f\colon V(G) \to [k] \cup \{*\}$ such that for every edge $uv \in E(G)$, either $f(u) \neq f(v)$ or $f(u) = f(v) = *$. That is, we allow a special flexible color $*$ in addition to the usual color set $[k]$, and adjacent vertices are permitted to share the color $*$. Given an extended k-coloring f of G, we define the set of *swappable colors* with respect to f as $S_f = \{c \in [k] \cup \{*\} \mid |f^{-1}(c)| = 1 \text{ or } (c = * \text{ and } f^{-1}(c) \neq \emptyset)\}$. Namely, a color c is swappable if it is used by exactly one vertex in G under f, or if $c = *$ and at least one vertex is assigned the color $*$ under f.

Our polynomial-time algorithm solves a generalized version of CRCS, which we call the Extended k-Coloring Reconfiguration (ECRCS) problem. In ECRCS, we are given a graph G and two extended k-colorings f_s and f_t of G, and the goal is to determine whether there exists a sequence of extended k-colorings that transforms f_s into f_t via color swaps. We use the terminology for ECRCS in the same way as for CRCS. Note that ECRCS generalizes CRCS in the following sense: for any instance (G, k, f_s, f_t) of CRCS, it holds that (G, k, f_s, f_t) is a yes-instance of CRCS if and only if it is a yes-instance of ECRCS. Furthermore, for any valid instance (G, k, f_s, f_t) of ECRCS, the initial and target colorings f_s and f_t must have the same set of swappable colors, i.e., $S_{f_s} = S_{f_t}$.

4.2.2 Polynomial-time Algorithm for Cographs

We now outline the strategy of our polynomial-time algorithm. Our approach is inspired by the polynomial-time algorithm for Token Sliding on cographs [22]. Indeed, an extended 1-coloring of a graph G naturally corresponds to an independent set of G, and thus our algorithm generalizes their result.

The algorithm proceeds recursively from the root to the leaves of the cotree T_G of the input cograph G, solving the ECRCS instance associated with each node t of T_G.

For a k-coloring f of G, let f^1 and f^2 denote the restrictions of f to $V(G_{t_1})$ and $V(G_{t_2})$, respectively. Suppose that we are given an instance (G, k, f_s, f_t) of ECRCS.

Leaf node. Let G be a cograph consisting of a single vertex v. Observe that (G, k, f_s, f_t) is a yes-instance of ECRCS if and only if $f_s(v) = f_t(v)$.

Union node. We consider the case where the root node of T_G is a union node with children t_1 and t_2. Since a union operation introduces no edges between $V(G_{t_1})$ and $V(G_{t_2})$, swaps involving vertices in $V(G_{t_1})$ and those in $V(G_{t_2})$ occur independently. Therefore, (G, k, f_s, f_t) is a yes-instance of ECRCS if and only if both $(G_{t_1}, k, f_s^1, f_t^1)$ and $(G_{t_2}, k, f_s^2, f_t^2)$ are yes-instances of ECRCS.

Join node. We now consider the case where the root node of T_G is a join node with children t_1 and t_2. Note that since t is a join node, all vertices in $V(G_{t_1})$ are adjacent to all vertices in $V(G_{t_2})$. Consequently, for any k-coloring of G, no color can appear in both f^1 and f^2.

We perform a case analysis based on whether any of the swappable color sets $S_{f_s^1}$, $S_{f_s^2}$, $S_{f_t^1}$, or $S_{f_t^2}$ is empty. We begin with the following observation.

▶ **Observation 17 (⋆).** *Let (G, k, f_s, f_t) be a yes-instance of* ECRCS*, where G is a cograph and the root node of T_G is a join node with children t_1 and t_2. Then, the following statements hold:*

1. *If at least one of $S_{f_s^1}$ or $S_{f_s^2}$ is empty, then no color swap between a vertex in $V(G_{t_1})$ and a vertex in $V(G_{t_2})$ occurs in any reconfiguration sequence from f_s to f_t.*
2. *$S_{f_s^1} = \emptyset$ if and only if $S_{f_t^1} = \emptyset$; similarly, $S_{f_s^2} = \emptyset$ if and only if $S_{f_t^2} = \emptyset$.*

We first consider the case where at least one of $S_{f_s^1}$ or $S_{f_s^2}$ is empty.

▶ **Lemma 18 (⋆).** *Let (G, k, f_s, f_t) be a valid instance of* ECRCS*, where G is a cograph and the root node of T_G is a join node with children t_1 and t_2. Suppose that at least one of $S_{f_s^1}$ or $S_{f_s^2}$ is empty. Then, (G, k, f_s, f_t) is a yes-instance of* ECRCS *if and only if both $(G_{t_1}, k, f_s^1, f_t^1)$ and $(G_{t_2}, k, f_s^2, f_t^2)$ are yes-instances of* ECRCS*.*

We then consider the remaining case. Let (G, k, f_s, f_t) be a valid instance of ECRCS, where G is a cograph and the root node of T_G is a join node with children t_1 and t_2. For each $j = 1, 2$, we define extended k-colorings $f_{s*}^j, f_{t*}^j \colon V(G_{t_j}) \to [k] \cup \{*\}$ as follows:

$$f_{s*}^j(v) = \begin{cases} * & \text{if } f_s^j(v) \in S_{f_s}, \\ f_s^j(v) & \text{otherwise} \end{cases}, \quad f_{t*}^j(v) = \begin{cases} * & \text{if } f_t^j(v) \in S_{f_t}, \\ f_t^j(v) & \text{otherwise} \end{cases}.$$

By construction, both f_{s*}^j and f_{t*}^j are extended k-colorings of G_{t_j}.

▶ **Lemma 19 (⋆).** *Let (G, k, f_s, f_t) be a valid instance of* ECRCS*, where G is a cograph and the root node of T_G is a join node with children t_1 and t_2. Suppose that both $S_{f_s^1}$ and $S_{f_s^2}$ are non-empty. Then, (G, k, f_s, f_t) is a yes-instance of* ECRCS *if and only if both $(G_{t_1}, k, f_{s*}^1, f_{t*}^1)$ and $(G_{t_2}, k, f_{s*}^2, f_{t*}^2)$ are yes-instances of* ECRCS*.*

Algorithm. The arguments for leaf, union, and join nodes naturally lead to a recursive algorithm for ECRCS on cographs. Since the set of swappable colors at each node of T_G and the corresponding extended colorings can be constructed in polynomial time, the overall algorithm runs in polynomial time. This concludes the proof of Theorem 16.

4.3 Split Graphs

Recall that CRCS is PSPACE-complete on split graphs, as shown in Theorem 2. In this subsection, we show a contrasting result: k-CRCS is solvable in polynomial time on split graphs when the number of colors k is a constant.

▶ **Theorem 20 (⋆).** CRCS *on split graphs admits a kernel with at most $k + 2k2^k$ vertices, where k is the number of colors of an input.*

Theorem 20 immediately leads to the following Corollary 21.

▶ **Corollary 21.** k-CRCS *can be solved in polynomial time for split graphs.*

5 Concluding Remarks

An intriguing direction is to determine the complexity of CRCS on graph classes that include paths. For example, does CRCS admit a polynomial-time algorithm on trees or on caterpillars? Another natural case is interval graphs, which are a subclass of chordal graphs where k-CRCS has been shown to be PSPACE-complete (Theorem 4). Since our algorithm for paths (Theorem 8) relies heavily on both the path structure and the restriction $k = 3$, it seems difficult to generalize our approach directly to broader classes. On the other hand, Token Sliding is known to be solvable in polynomial time on both trees [16] and interval graphs [4, 11]. Thus, as in the case of cographs (Theorem 16), it is conceivable that algorithms for CRCS could be derived from existing algorithms for Token Sliding.

References

1 Valentin Bartier, Nicolas Bousquet, Carl Feghali, Marc Heinrich, Benjamin Moore, and Théo Pierron. Recoloring planar graphs of girth at least five. *SIAM J. Discret. Math.*, 37(1):332–350, 2023. doi:10.1137/21M1463598.

2 Rémy Belmonte, Eun Jung Kim, Michael Lampis, Valia Mitsou, Yota Otachi, and Florian Sikora. Token sliding on split graphs. *Theory Comput. Syst.*, 65(4):662–686, 2021. doi:10.1007/S00224-020-09967-8.

3 Marthe Bonamy and Nicolas Bousquet. Recoloring bounded treewidth graphs. *Electron. Notes Discret. Math.*, 44:257–262, 2013. doi:10.1016/J.ENDM.2013.10.040.

4 Marthe Bonamy and Nicolas Bousquet. Token sliding on chordal graphs. In Hans L. Bodlaender and Gerhard J. Woeginger, editors, *Graph-Theoretic Concepts in Computer Science - 43rd International Workshop, WG 2017, Eindhoven, The Netherlands, June 21-23, 2017, Revised Selected Papers*, volume 10520 of *Lecture Notes in Computer Science*, pages 127–139. Springer, 2017. doi:10.1007/978-3-319-68705-6_10.

5 Marthe Bonamy, Nicolas Bousquet, Carl Feghali, and Matthew Johnson. On a conjecture of mohar concerning kempe equivalence of regular graphs. *J. Comb. Theory B*, 135:179–199, 2019. doi:10.1016/J.JCTB.2018.08.002.

6 Marthe Bonamy, Marc Heinrich, Takehiro Ito, Yusuke Kobayashi, Haruka Mizuta, Moritz Mühlenthaler, Akira Suzuki, and Kunihiro Wasa. Diameter of colorings under kempe changes. *Theor. Comput. Sci.*, 838:45–57, 2020. doi:10.1016/J.TCS.2020.05.033.

7 Édouard Bonnet, Tillmann Miltzow, and Pawel Rzazewski. Complexity of token swapping and its variants. *Algorithmica*, 80(9):2656–2682, 2018. doi:10.1007/S00453-017-0387-0.

8 Paul S. Bonsma and Luis Cereceda. Finding paths between graph colourings: PSPACE-completeness and superpolynomial distances. *Theor. Comput. Sci.*, 410(50):5215–5226, 2009. doi:10.1016/J.TCS.2009.08.023.

9 Paul S. Bonsma, Amer E. Mouawad, Naomi Nishimura, and Venkatesh Raman. The complexity of bounded length graph recoloring and CSP reconfiguration. In Marek Cygan and Pinar Heggernes, editors, *Parameterized and Exact Computation - 9th International Symposium, IPEC 2014, Wroclaw, Poland, September 10-12, 2014. Revised Selected Papers*, volume 8894 of *Lecture Notes in Computer Science*, pages 110–121. Springer, 2014. doi:10.1007/978-3-319-13524-3_10.

10 Nicolas Bousquet and Marc Heinrich. A polynomial version of Cereceda's conjecture. *J. Comb. Theory B*, 155:1–16, 2022. doi:10.1016/J.JCTB.2022.01.006.

11 Marcin Brianski, Stefan Felsner, Jedrzej Hodor, and Piotr Micek. Reconfiguring independent sets on interval graphs. In Filippo Bonchi and Simon J. Puglisi, editors, *46th International Symposium on Mathematical Foundations of Computer Science, MFCS 2021, August 23-27, 2021, Tallinn, Estonia*, volume 202 of *LIPIcs*, pages 23:1–23:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.MFCS.2021.23.

12 Charlie Carlson, Ewan Davies, Alexandra Kolla, and Will Perkins. Computational thresholds for the fixed-magnetization ising model. 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 1459–1472. ACM, 2022. doi:10.1145/3519935.3520003.

13 Luis Cereceda, Jan van den Heuvel, and Matthew Johnson. Connectedness of the graph of vertex-colourings. *Discret. Math.*, 308(5-6):913–919, 2008. doi:10.1016/J.DISC.2007.07.028.

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

15 Derek G. Corneil, H. Lerchs, and L. Stewart Burlingham. Complement reducible graphs. *Discret. Appl. Math.*, 3(3):163–174, 1981. doi:10.1016/0166-218X(81)90013-5.

16 Erik D. Demaine, Martin L. Demaine, Eli Fox-Epstein, Duc A. Hoang, Takehiro Ito, Hirotaka Ono, Yota Otachi, Ryuhei Uehara, and Takeshi Yamada. Linear-time algorithm for sliding tokens on trees. *Theor. Comput. Sci.*, 600:132–142, 2015. doi:10.1016/J.TCS.2015.07.037.

17 Tatsuhiko Hatanaka, Takehiro Ito, and Xiao Zhou. The coloring reconfiguration problem on specific graph classes. *IEICE Trans. Inf. Syst.*, 102-D(3):423–429, 2019. doi:10.1587/TRANSINF.2018FCP0005.

18 Robert A. Hearn and Erik D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theor. Comput. Sci.*, 343(1-2):72–96, 2005. doi:10.1016/j.tcs.2005.05.008.

19 Robert A. Hearn and Erik D. Demaine. *Games, puzzles and computation.* A K Peters, 2009.

20 Jan van den Heuvel. The complexity of change. In *Surveys in Combinatorics 2013*, volume 409 of *London Mathematical Society Lecture Note Series*, pages 127–160. Cambridge University Press, 2013. doi:10.1017/CBO9781139506748.005.

21 Matthew Johnson, Dieter Kratsch, Stefan Kratsch, Viresh Patel, and Daniël Paulusma. Finding shortest paths between graph colourings. *Algorithmica*, 75(2):295–321, 2016. doi:10.1007/S00453-015-0009-7.

22 Marcin Kaminski, Paul Medvedev, and Martin Milanic. Complexity of independent set reconfigurability problems. *Theor. Comput. Sci.*, 439:9–15, 2012. doi:10.1016/J.TCS.2012.03.004.

23 Kyozi Kawasaki. Diffusion constants near the critical point for time-dependent ising models. i. *Phys. Rev.*, 145:224–230, May 1966. doi:10.1103/PhysRev.145.224.

24 A. B. Kempe. On the geographical problem of the four colours. *American Journal of Mathematics*, 2(3):193–200, 1879.

25 Aiya Kuchukova, Marcus Pappik, Will Perkins, and Corrine Yap. Fast and slow mixing of the kawasaki dynamics on bounded-degree graphs. In Amit Kumar and Noga Ron-Zewi, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2024, August 28-30, 2024, London School of Economics, London, UK*, volume 317 of *LIPIcs*, pages 56:1–56:24. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.APPROX/RANDOM.2024.56.

26 Daniel Lokshtanov and Amer E. Mouawad. The complexity of independent set reconfiguration on bipartite graphs. *ACM Trans. Algorithms*, 15(1):7:1–7:19, 2019. doi:10.1145/3280825.

27 Bojan Mohar. *Kempe Equivalence of Colorings*, pages 287–297. Birkhäuser Basel, Basel, 2007. doi:10.1007/978-3-7643-7400-6_22.

28 Bojan Mohar and Jesús Salas. A new kempe invariant and the (non)-ergodicity of the Wang–Swendsen–Kotecký algorithm. *Journal of Physics A: Mathematical and Theoretical*, 42(22):225204, May 2009. doi:10.1088/1751-8113/42/22/225204.

29 Bojan Mohar and Jesús Salas. On the non-ergodicity of the Swendsen–Wang–Kotecký algorithm on the kagomé lattice. *Journal of Statistical Mechanics: Theory and Experiment*, 2010(05):P05016, May 2010. doi:10.1088/1742-5468/2010/05/P05016.

30 C. M. Mynhardt and S. Nasserasr. Reconfiguration of colourings and dominating sets in graphs: a survey. *CoRR*, abs/2003.05956, 2020. doi:10.48550/arXiv.2003.05956.

31 Naomi Nishimura. Introduction to reconfiguration. *Algorithms*, 11(4):52, 2018. doi:10.3390/A11040052.

32 Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970. doi:10.1016/S0022-0000(70)80006-X.

33 Alan D. Sokal. The multivariate tutte polynomial (alias potts model) for graphs and matroids. In Bridget S. Webb, editor, *Surveys in Combinatorics, 2005: Invited lectures from the Twentieth British Combinatorial Conference, Durham, UK, July 2005*, volume 327 of *London Mathematical Society Lecture Note Series*, pages 173–226. Cambridge University Press, 2005.

34 Marc Tedder, Derek G. Corneil, Michel Habib, and Christophe Paul. Simpler linear-time modular decomposition via recursive factorizing permutations. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games*, volume 5125 of *Lecture Notes in Computer Science*, pages 634–645. Springer, 2008. doi:10.1007/978-3-540-70575-8_52.

35 Tom C. van der Zanden. Parameterized complexity of graph constraint logic. In *Proceedings of IPEC 2015*, pages 282–293, 2015. doi:10.4230/LIPICS.IPEC.2015.282.

36 Eric Vigoda. Improved bounds for sampling colorings. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 51–59. IEEE Computer Society, 1999. doi:10.1109/SFFCS.1999.814577.

37 Katsuhisa Yamanaka, Erik D. Demaine, Takehiro Ito, Jun Kawahara, Masashi Kiyomi, Yoshio Okamoto, Toshiki Saitoh, Akira Suzuki, Kei Uchizawa, and Takeaki Uno. Swapping labeled tokens on graphs. *Theor. Comput. Sci.*, 586:81–94, 2015. doi:10.1016/J.TCS.2015.01.052.

38 Katsuhisa Yamanaka, Takashi Horiyama, J. Mark Keil, David G. Kirkpatrick, Yota Otachi, Toshiki Saitoh, Ryuhei Uehara, and Yushi Uno. Swapping colored tokens on graphs. *Theor. Comput. Sci.*, 729:1–10, 2018. doi:10.1016/J.TCS.2018.03.016.

Deterministic Local Problems in Radio Networks: On the Impact of Local Domination and a Bit of Advice

Pawel Garncarek
Institute of Computer Science, University of Wrocław, Poland

Tomasz Jurdzinski
Institute of Computer Science, University of Wrocław, Poland

Dariusz R. Kowalski
School of Computer and Cyber Sciences, Augusta University, GA, USA

Shay Kutten
Department of Data and Decision Sciences and the Grand Technion Energy Program, Technion, Haifa, Israel

Miguel A. Mosteiro
Computer Science Department, Pace University, New York, NY, USA

Abstract

Radio Networks (RN) is one of the fundamental models for network communication where nodes can broadcast messages locally but their simultaneous transmissions can interfere with each other at their shared neighbors. This work focuses on performing the very fundamental primitive of Local Broadcast, in spite of the interferences.

We investigate to what extent local knowledge, called *advice*, relating to the 2-local domination number γ_2 may speed up Local Broadcast. Specifically for each node and some dominating set, knowledge about some neighboring dominating node and the local number among the neighbors of that dominating node. We show that such advice is sufficient to build an efficient oblivious transmission schedule. Along those lines, we present three algorithms trading the level of adaptiveness (from oblivious to adaptive) for bits of advice per node (from $O(\log(\Delta\gamma_2))$ to 1). All our algorithms complete Local Broadcast in $\widetilde{O}(\Delta\gamma_2^2)$ rounds, where Δ is the maximum degree of the network.

On the side of lower bounds, we show that, for each quasi-adaptive deterministic Local Broadcast algorithm, there is some RN that requires $\Omega(\min\{(\min\{\Delta,\gamma_2\}/\log n)^2, n\})$ communication rounds, where n is the number of network nodes. In quasi-adaptive protocols nodes may stop executing once its computational task is completed. To the best of our knowledge, this is the first (nearly) quadratic Local Broadcast (same message for all neighbors) lower bound in the RN model. Our lower bound is stronger than previous works in multiple ways: *i*) it is nearly quadratically better than the best known general lower bound for this class of algorithms, *ii*) it applies to a wider class of algorithms than previous work for fully oblivious, *iii*) it achieves similar time lower bound than previous work proved for a much more demanding Local Broadcast where each node sends a *possibly different* message to each neighbor, and *iv*) it takes into account the local domination parameter γ_2.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Radio Networks, Local Broadcast, Distributed Deterministic Algorithms, Lower Bounds, Graph algorithms, Advice, Labeling Schemes, Local Domination

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.34

Funding *Pawel Garncarek*: Supported by the National Science Center, Poland (NCN), grant 35 2020/39/B/ST6/03288.
Tomasz Jurdzinski: Supported by the National Science Center, Poland (NCN), grant 35 2020/39/B/ST6/03288.

© Pawel Garncarek, Tomasz Jurdzinski, Dariusz R. Kowalski, Shay Kutten, and Miguel A. Mosteiro; licensed under Creative Commons License CC-BY 4.0

Shay Kutten: The work of this author was funded in part by grant 1346/22 from the Israeli Science 37 Foundation. His research was carried out within the framework of the Grand Technion Energy 38 Program (GTEP). A part of this work was done while with the Fraunhofer Darmstadt.
Miguel A. Mosteiro: Partially supported by Pace SRC grant and Kenan Fund.

1 Introduction

The Radio Networks (RN) model was introduced about 40 years ago by Chlamtac and Kutten [9] as a multi-hop generalization of a multiple-access channel [1, 41]. The model attracted a lot of attention over many years. A somewhat arbitrary list of examples includes [2, 3, 39, 12, 43, 25, 15, 23] among many others. One major difference between RN and, e.g., the CONGEST model of point-to-point communication [42], is that if a node v transmits when v's two-hop neighborhood does not, then *all* of v's neighbors receive copies of v's message, rather than just one of them. (The model is defined later.) Hence, the *Local Broadcast* task of sending messages to all the neighbors arises as a natural building block for RN, both local and global, see e.g., [40, 11]. For example, one can view some known algorithms for network-wide broadcast as using multiple applications of this building block of Local Broadcast. That is, in many works (e.g., [3, 13]) the broadcast proceeds by having nodes that already have received the broadcast message forward it by performing Local Broadcast.

The main challenge in RN is that simultaneous transmissions in the two-hop neighborhood of v may "collide" with v's message and prevent it from being received by some or all of v's neighbors. In this paper, we study the scheduling of nodes' transmissions that allows all the nodes to perform their Local Broadcast task in any network, overcoming their collisions. The nodes *do not know the network topology*, but may receive a few bits of advice, as discussed later.

Parameterizing RN graph topologies to measure time performance more accurately

Recently, Davies [15] suggested that the *independence number* of a graph (i.e., the size of a maximum independent set) can accurately parameterize time complexity of *global* problems, such as broadcast and leader election. More specifically, randomized algorithms in [15] have low time complexity as a function of the independence number of the network graph.

In the current paper, we address *local* tasks. Specifically, we consider *distributed deterministic algorithms* for the natural task of *Local Broadcast* and parameterize their time performance by a local version of the domination number of the graph, called 2-*local domination number* and denoted γ_2.[1] For a given radio network with topology graph $G = (V, E)$ and a set $B \subseteq V$, the Local Broadcast problem is to deliver a message from v to all neighbors of v, for each $v \in B$. Parameter γ_2 spans from constant to $O(\Delta^2)$, and in many popular classes, e.g., Unit Disc or Bounded Growth graphs, it is constant.

Preliminary intuitions about time complexity of deterministic Local Broadcast

Before stating our contributions formally, the following intuition may be interesting. Consider a node u that has Δ neighbors $v_1, v_2, ...v_\Delta$. In Local Broadcast, each such neighbor v_i needs to receive a message from each of its $O(\Delta)$ neighbors (other than u), and it has to be at some rounds r when u does *not* transmit in order to avoid simultaneous conflicting transmissions. Hence, only $O(\Delta^2)$ rounds are "forbidden" for u to transmit during Local Broadcast. The latter suggests that if u knew which rounds are "forbidden" for itself, it could perform Local Broadcast in $O(\Delta^2)$ time. On the other hand, Local Broadcast can be performed much faster

[1] Local domination is defined analogously to how local independence relates to independence, see [21].

Table 1 Our deterministic distributed RN Local Broadcast results in comparison with closest previous work. n is the number of network nodes, Δ is the maximum degree and γ_2 is the 2-local dominating number.

Section # or reference	Communication rounds	Protocol class	Bits of advice
[8]:Thm 13	$O(\Delta^2 \min\{\log n, \log^2_\Delta n\})$	oblivious, $\gamma_2 \geq 1$	none
[12]:Thm 1.6	$\Omega(\min\{\Delta^2 \log_\Delta n, n\})$	oblivious, $\gamma_2 \geq 1$	none
3.2	$O(\Delta\gamma_2^2 \log \gamma_2)$	oblivious	$O(\log(\Delta\gamma_2))$
3.3	$O(\Delta\gamma_2^2 \log n)$	quasi-adaptive	$O(\log \gamma_2)$
4.2	$\Omega\left(\min\left\{\left(\frac{\min\{\Delta,\gamma_2\}}{\log n}\right)^2, n\right\}\right)$	quasi-adaptive	none
3.4	$O(\Delta\gamma_2^2 \log^2 n)$	adaptive	1
[14]:Cor 11	$\Omega(\Delta \log n)$	any, $\gamma_2 \geq 1$	none

by a randomized algorithm [3, 14], with high probability. This suggests that for every network, there exists a transmission schedule in which some of the rounds forbidden for u because of neighbor v_i are the same as those forbidden because of some other neighbor v_j (for $j \neq i$). We can further infer then that a better deterministic Local Broadcast algorithm can be obtained if each node learns the whole topology (and is not computationally bounded). However, in this paper, we show that such a saving is not possible (up to a polylogarithmic factor) for oblivious and even *quasi-adaptive* algorithms (i.e., algorithms with predefined transmission schedules, but allowing nodes to stop based on received messages and channel feedback).

What topology knowledge, or advice, is sufficient to beat the Δ^2 bound?

The general question of how much of topology knowledge may help nodes in completing a distributed task faster, was heavily investigated in various forms, for example, in [17, 16, 18, 33, 23]. Inspired by [15], who parameterized time complexity of global broadcast by the size of a Maximal Independent Set, and by our lower bound involving the minimum local size of the Dominating Set (DS), we investigate to what extent some local knowledge about some DS may speed up Local Broadcast. In particular, the extra knowledge (also called advice, or, sometimes, a label) about some nearby node in the DS and the local number among the neighbors of that node is sufficient to build an efficient oblivious transmission schedule. Note that such advice consists of a logarithmic (in Δ, γ_2) number of bits per node, and could be further decreased down to 1 when adding some adaptiveness to the protocol.

1.1 Our Contributions

Our results are detailed and compared to previous work in Table 1.

Upper bounds. In Section 3 we present three Local Broadcast algorithms with advice, which are efficient (that is, $o(\Delta^2)$) for graphs with small 2-local domination number γ_2 (defined later). More precisely, all our algorithms complete Local Broadcast in $\widetilde{O}(\Delta\gamma_2^2)$ rounds (where notation $\widetilde{O}$ hides polylogarithmic factors) trading the size of advice for adaptivity.

The first algorithm (Section 3.2) needs $\log(\Delta\gamma_2^2) \in O(\log(\Delta\gamma_2))$ bits of advice, but it does not need any adaptivity (i.e., the algorithm is fully oblivious). In view of the best known lower bound $\Omega(\min\{\Delta^2 \log_\Delta n, n\})$ for oblivious algorithms without advice in [12], which works for any γ_2, it demonstrates that a logarithmic (in terms of Δ and γ_2) advice allows to beat this lower bound for networks with $\gamma_2 \in O(\sqrt{\Delta})$.

The second algorithm (Section 3.3) uses less advice of only $O(\log \gamma_2)$ bits, but needs a bit of adaptiveness in deciding to stop (i.e., the algorithm is quasi-adaptive). It is only one factor away from our new lower bound on quasi-adaptive algorithms without advice, described later.

The third algorithm (Section 3.4) requires only 1 bit of advice, but is fully adaptive.
All our algorithms *do not require* collision detection or acknowledgments built-in the model. They are also close to the obvious lower bound Δ (as a node can receive at most one message in a round, because of avoiding collisions), as long as γ_2 is small.

Lower bound. In Section 4 we present a lower bound of $\Omega\left(\min\left\{(\min\{\Delta,\gamma_2\}/\log n)^2, n\right\}\right)$ for quasi-adaptive deterministic algorithms accomplishing Local Broadcast in any n-node radio networks of node degree at most Δ and 2-local domination number γ_2 without any topology knowledge or any other kind of advice. For settings where either Δ or γ_2 are not restricted, our lower bound yields either $\Omega\left(\min\left\{(\gamma_2/\log n)^2, n\right\}\right)$ or $\Omega\left(\min\left\{(\Delta/\log n)^2, n\right\}\right)$ respectively. To the best of our knowledge, this is the first (nearly) quadratic Local Broadcast (same message for all neighbors) lower bound in the RN model. To prove it, we design and analyze a **hitting game on graphs**, in which one player tries to "hit" a matching hidden in a graph designed by other player, by asking queries (each query is a subset of nodes of the graph). See Section 4.1 and Theorem 17 for more details.

Our lower bound is stronger than previous results in three ways. It is nearly quadratically (in Δ) better than the best known general lower bound for this class of algorithms, mainly, $\Omega(\Delta \log n)$ [14].[2] Second, comparing to the $\Omega(\min\{\Delta^2 \log_\Delta n, n\})$ lower bound in [12] for fully oblivious algorithms, ours works for much wider class of quasi-adaptive algorithms and takes into account parameter γ_2, which as we described earlier, conveniently parametrizes time complexity of algorithms from the upper bound perspective. Third, our lower bound is also stronger than another similar lower bound of $\Omega(\Delta^2 \log n)$ in [14], which was proved for much more demanding communication task (under the name of Local Broadcast) – each node could send a *possibly different* message to each neighbor.

Other related work is overviewed in Appendix 1.1. Interesting extensions and open directions are discussed in Section 5.

A variety of *global* computational problems that are fundamental for distributed computing in communication networks has been studied under the RN model. *Local* broadcast on the other hand has been studied for randomized algorithms [3] and especially for particular cases of RN where, for instance, some (known) links are assumed to be unreliable and/or the solutions given are randomized (see [40, 26, 14] and references therein), or the topology is restricted e.g., to a complete graph. However, to the best of our knowledge, there is no previous work on *deterministic* Local Broadcast under the *general multi-hop* RN model. A non-comprehensive overview of related works, as well as lower bounds for other models that nevertheless applies to our context, are included below.

Historically, the problem of Local Broadcast was first considered in complete graphs, also called multiple-access channels or shared channels, where only a subset of nodes was active and ready to broadcast. Capetanakis [7], Hayes [31], and Tsybakov and Mikhailov [44] independently proposed a deterministic adaptive tree-like algorithm in the model with collision detection, working in $O(\Delta \log(n/\Delta))$ communication rounds, where Δ is the number of active

[2] The lower bound $\Omega(\Delta \log n)$ in [14] works also for adaptive, and even randomized, protocols, however it does not take into account parameter γ_2.

nodes. The same time complexity can be achieved even by quasi-adaptive algorithms [36, 38] in a channel with acknowledgments but without collision detection. Clementi, Monti, and Silvestri [12] showed a matching $\Omega(\Delta \log(n/\Delta))$ lower bound for such a setting, which also holds for adaptive algorithms. As for the setting with collision detection, the best lower bound $\Omega(\Delta \log n / \log \Delta)$ was given by Greenberg and Winograd [30], leaving the logarithmic gap open for 40 years now.

In arbitrary RN, the best previously known deterministic Local Broadcast algorithm required time $O(\Delta^2 \min\{\log n, \log_\Delta^2 n\})$ by Cheraghchi and Nakos [8]. It (roughly) logarithmically improved upon the algorithm by Clementi, Monti and Silvestri [12], based on superimposed codes introduced by Kautz and Singleton [35]. Both these algorithms are oblivious, and as such do not need collision detection or acknowledgments. The complexity of randomized Local Broadcast in arbitrary networks of node degree Δ in the related Beeping Networks models matches the general lower bound and is $\Theta(\Delta \log n)$, as shown in [14].

There is a rich literature about distributed algorithms and even non-distributed online algorithms with advice and/or with informative labels, e.g., [22, 24, 34, 45, 37, 5].

2 Model, Notation, Problems, and Protocols

We use a standard description of the Radio Networks (RN) model, formulated 40 years ago [9]. Consider a communication network formed by n devices with communication and computation capabilities, called **nodes**. Each node has a unique **ID** from the range $[1, n^c]$, for some constant $c \geq 1$.[3] Nodes communicate by sending **messages** among them. A message may be lost in cases specific to the nature of radio networks, defined later. A message is composed of a binary sequence containing the source node ID, the destination node ID (if applicable), and the specific information to be sent.

Network topology and communication. If a message from the source node is not lost, and it is received by the destination node, we say that the message was **delivered**. Each pair of nodes that are able to communicate directly (i.e., without relaying communication through other nodes) are said to be connected by a communication **link** and are called **neighbors**. We assume that links are **symmetric**, i.e., messages can be sent in both directions. For each node $v \in V$, the set of neighbors of v is called its **neighborhood**, denoted as $N(v)$. The network topology defined by the communication links is modeled by an undirected graph $G = (V, E)$ where V is the set of nodes/vertices and E is the set of links/edges. The network $G = (V, E)$ is **connected** if E is such that for every pair of nodes $u, v \in V$, there is a path of links connecting u and v. The maximum node degree in G is denoted by $\Delta = \max_{v \in V} |N(v)|$; we assume that nodes know a linear upper estimate of Δ.

We assume that time is slotted in **rounds** of communication such that if there is a link between nodes u and v and a message is sent by u in some round, the message is either delivered to v in the same round or it is lost (in the cases detailed later). In both cases, we say that v **transmits** in that round. If u does not transmit in some round, we say that it **listens** in that round, meaning that it waits for the reception of the transmission of a neighbor, an event that may or may not happen. All the nodes start running protocols simultaneously, i.e., the network is **synchronous**. We assume that computations take negligible time with respect to communication. Thus, we measure algorithm performance in rounds.

[3] The availability of identifiers is essential in order to break symmetry in deterministic RN protocols, as pointed out in previous works, e.g., [28].

Transmission result. In a round when a node v listens, it may either receive a message - this is the case that exactly one of its neighbors transmits in that round. Or, it may also hear **silence** - this is the case that either none of its neighbors transmits or some set $\{u_1, u_2, ...u_k\}$) of neighbors of v, for $k > 1$ does. In the second case, we say that the messages of $\{u_1, u_2, ...u_k\}$ **collided** at v. In a round, a node v that transmits any message does not receive any message sent by any of its neighbors $u \in N(v)$. We say that u's message collided with v's message at v. If a node u transmits and its neighbor v does not receive the message (in the cases defined above), we say that u's message is lost on the way to v. This does not mean that the other neighbors of u do not receive the message from the transmission of u.

Networks with collision detection (CD). We may or may not assume a third feedback a listening node v may receive (instead of either receiving a message or hearing silence). In a radio network equipped with CD, if more than one of v's neighbors transmit in some round, v hears **noise** rather than silence. Obviously, an algorithm that does not use CD can function with the same complexities in networks with CD.

Advice. Some of the results presented in this paper are in the framework of algorithms with **advice** or, alternatively, with **informative labels**. In this paradigm that has recently got growing attention, an oracle knowing the network provides binary strings to nodes before the beginning of a computation. A distributed algorithm uses this advice to solve the particular problem. The required size of advice (the maximum length of the strings) can be considered as a measure of the difficulty of the problem (in terms of "how much topology knowledge is needed"). Two variations are studied in the literature: either the binary string given to nodes is the same for all of them [27] or, as we do, different strings may be given to different nodes (as in e.g. [19, 28, 6, 29]).

Preliminaries and Specification of Local Broadcast. The following notations are used for the topology graph $G = (V, E)$ of a radio network. A path between two nodes $u, v \in V$ is a sequence of adjacent links without cycles connecting u and v. If the shortest path from u to v contains x links, we say that v is within x **hops** from u and vice versa, and we also say that the **distance** between u and v is x. Let k be a natural number. The ***k*-neighborhood** of a node v is the set of all nodes within k hops from v, including v itself.

A **distance k coloring of $S \subseteq V$** is a color assignment such that every node $v \in S$ receives a color such that any two nodes $u, w \in S$ that received the same color cannot be within k distance of each other. E.g., the standard coloring is distance 1 coloring of V.

A (distance-1) **dominating set** of a graph $G = (V, E)$ is a set $S \subseteq V$ such that every node of $V \setminus S$ is at distance 1 from some node in S. The (global) **domination number**, $\gamma(G)$, of G is the minimum cardinality of any dominating set of G [32]. We will use the following local version of the (distance-1) domination number. Our definition is consistent with the local version of independence characterizations [21]. Let $\mathcal{S}$ be the set of dominating sets of G. For any integer $k \geq 1$ and set $S \in \mathcal{S}$, the **k-local domination number $\gamma_k(G, v, S)$ of node $v \in V$** is the number of nodes in S within distance k of v in G. We also define a **k-local domination number $\gamma_k(G, V', S) = \max_{v \in V'} \gamma_k(G, v, S)$ of a set of nodes $V' \subseteq V$**, and the **k-local domination number $\gamma_k(G) = \min_{S \in \mathcal{S}} \gamma_k(G, S)$ of the graph G**, where $\gamma_k(G, S) = \gamma_k(G, V, S)$.

To help better understand the k-local domination number, we provide Observations 1, 2 and 3 below. Their proofs, including graph examples, will be provided in the full version of this paper.

▶ **Observation 1.** *For every Δ, there exists a graph G with degree Δ such that the 2-local domination number $\gamma_2(G) \in \Theta(\Delta^2)$.*

▶ **Observation 2.** *Consider any connected graph $G = (V, E)$ where $V \neq \emptyset$. The 2-local domination number $\gamma_2(G) = 1$ if and only if there exists a dominating set DS on graph G such that $|DS| = 1$.*

Note that a dominating set of size $|DS| = 1$ can only exist in some graphs with diameter $D \leq 2$.

The above lower bound on the 2-local dominating number $\gamma_2(G)$ may be unsatisfactory, since it only applies for small graphs where the dominating set has only one node. However, one can note that the dominating number can be as low as 2 in many large and complex graphs.

▶ **Observation 3.** *The 2-local domination number $\gamma_2(G)$ can be as low as 2 in some graphs with arbitrarily large n, Δ, D and $|DS|$, where D is the diameter of the graph G and DS is some dominating set.*

The formal definition of the Local Broadcast problem is as follows.

▶ **Definition 4** (Local Broadcast Problem)**.** *Given a radio network with topology graph $G = (V, E)$, where each node $v \in B$, for some set $B \subseteq V$, holds a message m_v, the problem is solved once m_v for every $v \in B$ is delivered to every node in $N(v)$.*

It is enough for us to consider only the case that $B = V$. A lower bound for this case is an existential lower bound for a general B. Similarly, it is easy to show that an upper bound for the case that $B = V$ holds for a smaller B too. Hence, in this paper, $B = V$.

Protocol Classes. We give now a precise definition of the classes of algorithms considered in this work. The following will be used. In the context of a communication protocol for radio networks with set of nodes V, let a **schedule of transmissions** be a Boolean matrix $\left[b_{v,r}\right]_{v\in V, r\in[1,\lambda]}$, where $b_{v,r}$ indicates whether node v transmits in round r or not. We say that λ is the **length** of the schedule. We consider two classes of deterministic Local Broadcast protocols: oblivious and quasi-adaptive.

An **oblivious protocol** is a schedule of transmissions $\left[b_{v,r}\right]_{v\in V, r\in[1,\lambda]}$ to be used by the nodes of the network. The schedule is pre-defined before execution, and it does not change until round λ, when all the nodes stop.

A **quasi-adaptive protocol** is a schedule of transmissions $\left[b_{v,r}\right]_{v\in V, r\in[1,\lambda]}$ to be used by the nodes of the network. The schedule is predefined before execution, but any given node may independently stop its execution in any round $1, \ldots, \lambda$ after its computational task has been completed.

Quasi-adaptive protocols in multi-hop networks have been considered before, e.g. in [38] (Section 5). In such protocol nodes use channel feedback to stop, hence, they are not strictly oblivious (in the sense of lack of adaptive switch-off, as considered e.g., in [13]). However, some of their components are oblivious (based on fixed selectors). Also, many randomized algorithms are quasi-adaptive, in the sense that they transmit according to fixed distribution of schedules but stop immediately after getting acknowledgment of successful transmission(s) - the reader may find some examples of backoff-based quasi-adaptive algorithms e.g., in [38].

An **adaptive protocol** is a protocol in which each node v determines whether to transmit or listen in a given round r based on the ID of v and on the feedback from the radio channel for all rounds $r' < r$, where the feedback for a given round r' is either silence if v does not receive any message in the rounds r' or the content of the received message otherwise.

3 Algorithms with Advice to Beat the Lower Bound

In this section, we present three algorithms, trading the level of adaptiveness (from oblivious to adaptive) for bits of advice per node (from $O(\log(\Delta\gamma_2))$ to 1). All of them complete in $\widetilde{O}(\Delta\gamma_2^2)$ rounds. The assumptions that follow will be used.

Suppose there is a fixed dominating set (DS), known to all the nodes. Each node not in the DS is assigned to one of its neighbors in this DS. A node in the DS is assigned to itself. Moreover, each node $v \in DS$ has a unique numbering of nodes assigned to itself, $1, \ldots, i$, where i is the number of nodes assigned to v. Note that $i \leq \Delta + 1$, because the assignment is of nodes neighboring v plus the assignment of v to itself.

Initially, the above knowledge will come from an oracle (see Section 3.2). In Section 3.4, we show how to find all the above assignments but assuming a given marking of the nodes in the DS, which can be accomplished with a single bit of advice from an oracle.

Our algorithms will also use the combinatorial structures called **selectors**, **avoiding selectors** and **strong selectors**, presented below in Section 3.1. In particular, specific selectors from that section are integrated in our algorithms.

3.1 Preliminaries: Selectors

▶ **Definition 5** (Selectors). *A family $\mathcal{F}$ of subsets of $[n]$ is called an* (n, k)-selector *of length $|\mathcal{F}|$, if for every non-empty subset $S \subseteq [n]$ such that $|S| \leq k$, there exists a set $F \in \mathcal{F}$ such that $|F \cap S| = 1$.*

▶ **Theorem 6** ([13]). *For any $n > k \geq 2$, there exists an (n, k)-selector of length $O(k \log(n/k))$.*

▶ **Definition 7** (Avoiding selectors). *A family $\mathcal{F}$ of subsets of $[n]$ is called an* (n, k, ℓ)-avoiding selector *of length $|\mathcal{F}|$, where $1 \leq \ell < k \leq n$, if for every non-empty subset $S \subseteq [n]$ such that $|S| \leq k$ and for any subset $R \subseteq S$ of size at most ℓ, there is an element $a \in S \setminus R$ for which there exists a set $F \in \mathcal{F}$ such that $F \cap S = \{a\}$.*

The following fact follows directly from Definition 7, see also [4, 10].

▶ **Fact 8.** *Suppose we are given an (n, k, ℓ)-avoiding selector $\mathcal{F}$ and a set S of size at most k. Then, the number of elements in S not "selected" by selector $\mathcal{F}$ (i.e., for which there is no set in the selector that intersects S on such singleton element) is smaller than $k - \ell$.*

▶ **Theorem 9** ([4, 10]). *There exists an (n, k, ℓ)-avoiding selector of length $O\left(\frac{k^2}{k-\ell} \log n\right)$.*

▶ **Definition 10** (Strong selectors). *A family $\mathcal{F}$ of subsets of $[n]$ is called a* (n, k)-strong selector *of length $|\mathcal{F}|$, if for every non-empty subset $S \subseteq [n]$ such that $|S| \leq k$, for every element $a \in S$, there exists a set $F \in \mathcal{F}$ such that $F \cap S = \{a\}$.*

▶ **Theorem 11** ([20, 35]). *For any $n \geq 3$ and for $k \geq 2$, there exists an (n, k)-strong selector of length $O(\min\{n, k^2 \log n\})$.*

Note that selectors do not depend on the network, so they can be computed in a non-distributed way in advance and integrated into our algorithms.

3.2 An Oblivious Algorithm LB_LAd with $O(\log(\Delta\gamma_2))$ Advice

At the beginning, the oracle determines a dominating set DS which minimizes[4] the value $\gamma_2 = \gamma_2(G, DS)$. Then, the oracle chooses a distance 4 coloring of DS in G such that the number of colors is $O(\gamma_2^2)$. Note that such a coloring always exists since, by the definition of γ_2, the number of nodes $u \in DS$ in distance ≤ 2 from a node $v \in V$ is at most γ_2. I.e., one can greedily color DS nodes using $\gamma_2^2 + 1$ colors. Next, the oracle assigns each $u \in V$ to its neighbor from DS and, for each $v \in DS$, provides unique labels to the nodes assigned to v from the range $[\Delta]$.

Each node v receives from the oracle three numbers: the color of the DS node, say w, to which v has been assigned by the oracle, the label of v among the nodes assigned to w, and the value of γ_2. We assume here a DS node is always assigned to itself with the special label 0 indicating that it is a member of DS. It could be easily seen that the length of advice is $O(\log(\Delta\gamma_2))$ in each node.

Let $\mathcal{F}$ be a (γ_2^2, γ_2)-strong selector. We could assume that the number of sets in $\mathcal{F}$ is $O(\gamma_2^2 \log \gamma_2)$, see Theorem 11.

The algorithm is split into phases, each of $\Delta + 1$ rounds. A node v is *active* in phase i if the color of the DS node w to which v is assigned is in the ith set of $\mathcal{F}$; otherwise, v is *passive*. An active node v transmits its message, combined with its ID and the color of the assigned DS node w, in round x of the phase and is silent in all other rounds, where x is the label of v among nodes assigned to w (recall that this number is provided to v at the beginning of the computation as part of the advice). See Algorithm 1 for a pseudo-code.

Algorithm 1 LB_LAd– Local Broadcast algorithm with $O(\log(\Delta\gamma_2))$ Advice for a node v.

```
Let F = {S_1, ..., S_ℓ} be the fixed (shared between all nodes) (γ_2^2, γ_2)-strong selector, where ℓ = |F|
Receive (from oracle) color c of DS node w, in some distance 4 coloring, that v will be assigned to
Receive from the oracle the number x assigned to v
for phase = 1, ..., ℓ do
    for round = 1, ..., Δ + 1 do
        if w ∈ S_phase and round = x then
            Transmit the message, ID of v and the color of w
        else
            Remain silent
        end if
    end for
end for
```

▶ **Theorem 12.** *Algorithm LB_LAd is a deterministic distributed oblivious protocol performing Local Broadcast in $O(\Delta\gamma_2^2 \log \gamma_2)$ communication rounds. It uses $O(\log(\Delta\gamma_2))$ bits of advice per node, and does not require collision detection or acknowledgments built-in the model.*

Proof. Consider any two neighboring nodes v, u. We prove that v delivers its message successfully to u by the algorithm. Let w be the DS node to which v is assigned and let $DS_2(u)$ be the set of DS nodes of distance at most 2 from u. By definition, $|DS_2(u)| \leq \gamma_2$.

[4] The algorithm works with any DS, but the round complexity depends on $\gamma_2(G, DS)$.

Note that each neighbor of u is assigned to some node in $DS_2(u)$, and is using the assigned node's color to determine, based of the corresponding set in $\mathcal{F}$, whether to be active or passive in a phase. Since $\mathcal{F}$ is a (γ_2^2, γ_2)-strong selector, there is a set in $\mathcal{F}$ containing w but no other node in $DS_2(u)$. Let it be a set i, which corresponds to being active or not in phase i. It means that only neighbors of u that are assigned to w are active in phase i. In this phase, they perform a round-robin according to their numbering (each of them knows its number, as it has been provided as part of the advice). Hence, there is no interference between these nodes at node u while recalling that all other neighbors of u are passive in phase i. It implies that node v, which is among neighbors of u assigned to w, performs transmission in a unique round of phase i and is successfully received by u.

The number of rounds is obviously upper bounded by the number of sets in the strong selector $\mathcal{F}$, multiplied by the length of each phase $\Delta + 1$. ◀

3.3 A Quasi-adaptive Algorithm with $O(\log \gamma_2)$ Advice

At the beginning, the oracle determines a dominating set DS which minimizes[5] the value $\gamma_2 = \gamma_2(G, DS)$. Then, it chooses a distance 4 coloring of DS in G such that the number of colors is $O(\gamma_2^2)$. As explained in Section 3.2, such a coloring exists. Next, the oracle assigns each $u \in V$ to its neighbor from DS. Contrary to Section 3.2, labels are not provided.

Each node v receives two numbers from the oracle : the color of the DS node, say w, to which v has been assigned by the oracle and the value of γ_2. We assume here a DS node is always assigned to itself with the special label 0 indicating that it is a member of DS. It could be easily seen that the length of advice is $O(\log \gamma_2)$ in each node.

Let $\mathcal{F}_i$ be a $(n, \Delta/2^{i-1}, \Delta/2^i)$-avoiding selector, $\mathcal{F}_i = (S_1^{(i)}, \ldots, S_{\ell_i}^{(i)})$. The number of sets in $\mathcal{F}_i$ is $O(\Delta/2^{i-1} \log n)$, see Theorem 9.

The algorithm is split into $\log \Delta$ phases. The ith phase consists of $|\mathcal{F}_i| = \ell_i$ stages. Finally, each stage has $\gamma_2^2 + 1$ blocks, each block has 2 rounds. A node $v \notin DS$ is *active* in stage j of a phase i if it was not switched off earlier and its ID belongs to $S_j^{(i)}$. An active node v transmits its message, combined with its ID in the Round 1 of the block k such that k is the color of the assigned DS node w. If a node $v \in DS$ receives a message in Round 1 of the block k such that the color of v is equal to k then v sends the message back in Round 2 of that block. If $v \notin DS$ sends a message in Round 1 of a block and receives its message back in Round 2 of that block, it switches off for the remaining part of the algorithm. See Algorithm 2 for a pseudo-code.

▶ **Theorem 13.** *Algorithm LB_quasi (Algorithm 2) is a deterministic distributed quasi-adaptive protocol performing Local Broadcast in $O(\Delta\gamma_2^2 \log n)$ communication rounds. It uses $O(\log \gamma_2)$ bits of advice per node, and does not require collision detection or acknowledgments built-in the model.*

Proof. Consider any two neighboring nodes v, u. Assume that v switches off after some block determined by the current values of the variables *phase*, *stage* and *block*. We claim that u receives the message from v in Round 1 of the given block which we will call the current block. Let w be the DS node to which v is assigned. We denote such node by $\mathrm{DS}(v)$, i.e., $\mathrm{DS}(v) = w$ according to the previous sentence. Thus $col(v) = col(w) = block$ and the colors of all nodes from DS in distance at most 4 from w are different from *block*. As a result, the color of each node x in distance at most 3 from w such that $\mathrm{DS}(x) \neq w$ (i.e., x is not

[5] The algorithm works with any DS, but the round complexity depends on $\gamma_2(G, DS)$.

Algorithm 2 LB_quasi – Local Broadcast algorithm with $O(\log \gamma_2)$ Advice for a node v.

Let $\mathcal{F}_i = \{S_1^{(i)}, \ldots, S_{\ell_i}^{(i)}\}$ be a fixed (shared between all nodes) $(n, \Delta/2^{i-1}, \Delta/2^i)$-avoiding selector, where $\ell_i = |\mathcal{F}_i|$, for $i = 0, 1, \ldots, \log \Delta$.
Receive from the oracle the color $\mathrm{col}(v) = c$ of DS node w such that v is assigned to w
for *phase* $= 1, \ldots, \log \Delta$ **do**
    for *stage* $= 1, \ldots, \ell_{phase}$ **do**
        for *block* $= 1, \ldots, \gamma_2^2 + 1$ **do**
            Round 1:
            if $v \in S_{stage}^{(phase)}$ and $\mathrm{col}(v) =$ *block* and $v \notin DS$ **then**
                Transmit the message and the ID of v
            else
                Remain silent
            end if
            Round 2:
            if $\mathrm{col}(v) =$ *block* and $v \in DS$ **then**
                if v received a message in the previous round **then**
                    send the received message back
                else
                    Send a dummy message
                end if
            end if
            if v sent a message in Round 1 and received it back in Round 2 **then**
                v switches off
            end if
        end for
    end for
end for

assigned to w) is different from *block*. Since v is a neighbor of w, the color of each node in distance at most 2 from v such that $\mathrm{DS}(x) \neq \mathrm{DS}(v) = w$ is different from *block* as well. This fact implies that the only reason for which the message from v is not received by its neighbor u in Round 1 of *block* is that another neighbor of $\mathrm{DS}(v) = w$ transmits in that round. Then, however, there is also a collision at w in Round 1 and w will not transmit any message in Round 2 which prevents v from switching of after the current block. But this fact in turn would contradict the assumption that v switches off after the *block*.

So already know that if v eventually switches off during an execution of the algorithm, it performs its local broadcast task successfully. Thus, it remains to prove that actually each node switches off during an execution of Algorithm 2. The above considerations imply that, in order to guarantee that v switches off is some block, it is sufficient that it is the only element of the intersection of $S_{stage}^{(phase)}$ and the set of such nodes w that $\mathrm{DS}(v)$. As the set of nodes assigned to a particular element w of DS is included in the set of neighbors of w, its initial size is at most Δ. Let X_{phase} be the set of nodes assigned to w which were not switched before the given *phase*. Note that each node x selected by S_{stage}^{phase} as the unique element among X_{phase} (i.e., such that $X_{phase} \cap S_{stage}^{phase} = \{x\}$) is switched off in the given phase. With this observation one can easily prove inductively that the size of X_{phase} is at most $\Delta/2^{phase-1}$ and $\mathcal{F}_{phase}$ selects $\Delta/2^{phase}$ of them. Thus eventually, after $\log \Delta$ phases all nodes are switched off.

The number of rounds of the given $phase \in [1, \log \Delta]$ is $O(\ell_{phase} \cdot \gamma_2^2) = O(\Delta/2^{i-1} \cdot \gamma_2^2 \log n)$. And the time of the whole algorithm's execution is $O(\sum_{i=1}^{\log \Delta} \Delta/2^{i-1} \cdot \gamma_2^2 \log n) = O(\Delta \gamma_2^2 \log n)$. ◀

3.4 Adaptivity Reduces the Need of Advice to One Bit

At the beginning, the oracle determines a dominating set DS which minimizes[6] the value $\gamma_2 = \gamma_2(G, DS)$. Then each node only gets one bit of advice: whether it is in the DS or not. The procedures described next allow the computing of both the assignment and the local numbering at DS nodes by deterministic distributed algorithms. Knowing the assigned DS node and a unique local number in that node (in $\{1, \ldots, \Delta + 1\}$), the main algorithm uses a (n, γ_2)-strong selector $\mathcal{F}$ of length $|\mathcal{F}| \in O(\gamma_2^2 \log n)$ as follows: a node transmits in round $t = (i-1) \cdot (\Delta + 1) + j$, where $i \in [|\mathcal{F}|]$ and $j \in \{1, \ldots, \Delta + 1\}$, if its assigned DS node is in the set F_i of selector $\mathcal{F}$ and its local number is j. The pseudocode is in Algorithm 3.

Algorithm 3 Local broadcast after assignment of DS nodes and numbering – Algorithm for node v.

```
Let F = {S_1, ..., S_ℓ} be the fixed shared (n, γ_2)-strong selector, where ℓ = |F|
for i = 1, ..., ℓ do
    for j = 1, ..., Δ + 1 do
        DS_v ← the DS node assigned to v
        if DS_v ∈ F_i and the local number of v is equal to j then
            v transmits
        end if
    end for
end for
```

We continue with describing the procedures of assigning DS node and local numbering.

DS-node Assignment Procedure

DS nodes use an (n, γ_2)-selector $\mathcal{F} = \{S_1, S_2, \ldots, S_\ell\}$, where $\ell = |\mathcal{F}|$. In round i, DS nodes that are in the set S_i transmit – intuitively, they try to deliver their IDs to neighboring nodes. Each non-DS node sets the sender of the first message received as its assigned DS node. For completeness, each DS node sets itself as its own assigned DS node. This procedure is clearly oblivious. For detailed pseudocode, see Algorithm 4.

[6] The algorithm works with any DS, but the round complexity depends on $\gamma_2(G, DS)$.

Algorithm 4 Assignment of DS nodes – Algorithm for node v.

Let $\mathcal{F} = \{S_1, \ldots, S_\ell\}$ be the fixed shared (n, γ_2)-selector, where $\ell = |\mathcal{F}|$ (n, Δ)-strong selector, where $k = |\mathcal{F}'|$
$A \leftarrow \emptyset$ ▷ denotes the set of nodes assigned to v
if $v \in DS$ **then**
    $assignment \leftarrow v$
end if
for $round = 1, \ldots, \ell$ **do** ▷ first part of the algorithm
    if $v \in S_{round}$ and $v \in DS$ **then**
        Transmit ID of v
    end if
    if received a transmission with ID of a node u and $assignment = null$ **then**
        $assignment \leftarrow u$
    end if
end for

▶ **Lemma 14.** *The DS-node Assignment procedure terminates in $O(\gamma_2 \log(n/\gamma_2) + \Delta \log n)$ communication rounds. At the end, each node v knows the ID of the DS node it is assigned to.*

Proof. From Theorem 6, the length of (n, γ_2)-selector, and thus also the number of communication rounds in the whole procedure, is $O(\gamma_2 \log(n/\gamma_2))$.

From Definition 5, for any non-DS node v there will be a round in the procedure such that v hears exactly one transmission. The transmissions only come from the DS nodes. Thus, any non-DS node v will be assigned to some DS node. Recall that any DS node is assigned to itself, by default. ◀

Numbering Procedure

This procedure proceeds in $1 + \log \Delta$ stages. In each stage, every DS node assigns numbers to half of its remaining unnumbered assigned nodes, about which it learns in this stage. In the i-th stage, all the nodes use a $(n, \Delta/2^{i-1}, \Delta/2^i)$-avoiding selector $\mathcal{F}_i$, for $i = 1, \ldots, 1 + \log \Delta$. Intuitively, the second parameter of the selector (equal to $\Delta/2^{i-1}$) is the upper bound on the number of neighbors assigned to a DS node by the preceding DS-node Assignment Procedure, which are possible unknown to the DS node, and thus locally unnumbered, at the start of the stage. The third parameter of the selector (equal to $\Delta/2^i$) is the upper bound on the number of assigned neighbors that may remain unnumbered after the stage.

For each set S in $\mathcal{F}_i$, there are two phases: in the first phase, the unnumbered nodes selected by S (i.e., contained in S) transmit their IDs with ID of their DS-assigned node (known prior the procedure, because of the preceding DS-node Assignment Procedure, see also Lemma 14). If a DS node v receives an ID of a node u with the information that u is assigned to v, then, in the second phase, v transmits a response (also called an acknowledgment), composed of a unique number assignment and IDs of v and u. This (locally) unique number starts with 2 (number 1 is reserved for the DS node itself), and is incremented by 1 after each such activity of the DS node v.

The first phase only uses a single communication round, though not all messages may be successfully received. The second phase is accomplished by using the (n, γ_2)-strong selector $\mathcal{G} = \{T_1, T_2, \ldots, T_\ell\}$, where $\ell = |\mathcal{G}|$. Hence, the second phase takes ℓ communication rounds – each DS node v that is supposed to send an acknowledgment (with content described earlier) does so in rounds i such that $v \in T_i$. For detail pseudocode, see Algorithm 5.

Algorithm 5 Numbering nodes – Algorithm for node v.

Let $\mathcal{F}_i$, for $i = 1, \ldots, 1 + \log \Delta$, be the fixed shared $(n, \Delta/2^{i-1}, \Delta/2^i)$-avoiding selector
Let $\mathcal{G} = \{T_1, \ldots, T_\ell\}$ be the fixed shared (n, α_2)-strong selector, where $\ell = |\mathcal{G}|$
Let *assignment* denote the DS node that v is assigned to
Let A denote the set of nodes assigned to v ▷ only used if $v \in DS$
number $\leftarrow$ *null* ▷ denotes the number assigned to v
if $v \in DS$ **then**
    number $\leftarrow 1$
end if
current $\leftarrow 2$ ▷ denotes the next number that will be given to a neighbor
for *stage* $= 1, \ldots, 1 + \log \Delta$ **do**
    for $S \in \mathcal{F}_{stage}$ **do**
        received $\leftarrow$ *null* ▷ ID of a node that v will respond to in the second phase
        round $\leftarrow 1$ ▷ denotes the round number of processing S
        if *number* $=$ *null* and $v \notin DS$ and $v \in S$ **then**
            Transmit ID of v
        end if
        if $v \in DS$ and received ID of a node u and $u \in A$ **then**
            received $\leftarrow u$
        end if
        for *round* $= 2, \ldots, \ell + 1$ **do** ▷ second phase
            if $v \in DS$ and *received* $\neq$ *null* and $v \in T_{round-1}$ **then**
                Transmit (*current*, ID of v, ID of *received*)
                current $\leftarrow$ *current* $+ 1$
            end if
            if received (a, b, c) and $b =$ *assignment* and $c = v$ **then**
                number $\leftarrow a$
            end if
        end for
    end for
end for

▶ **Lemma 15.** *The Numbering procedure finishes in $O(\Delta\gamma_2^2 \log n)$ communication rounds. Consider any node v and a node u such that v is assigned to u. The Numbering procedure guarantees that v knows a unique number $1, \ldots, \Delta + 1$ among all the nodes assigned to u.*

Proof. For each set S in $\mathcal{F}_i$ for any i, phase one takes 1 round and phase two takes $|\mathcal{G}|$ rounds. According to Theorem 11, there are $1 + |\mathcal{G}| = O(\gamma_2^2 \log n)$ communication rounds for each S and i. There are $|\mathcal{F}_i|$ such sets S in a stage i. According to Theorem 9, there exists a constant c such that $|\mathcal{F}_i| \leq c\Delta/2^i \log n$ for each i. Therefore, $\sum_{i=1}^{\log \Delta} |\mathcal{F}_i| \leq 2c\Delta \log n = O(\Delta \log n)$. Hence, there are $O(\Delta\gamma_2^2 \log^2 n)$ communication rounds in total.

When we look at all the phases of a single stage i, then, from Fact 8, there are at most $\Delta/2^{i-1}$ non-DS assigned neighbors that failed to transmit their IDs to their assigned DS node.

In phase two, every acknowledgment is received: From Definition 10, there is a round j such that the acknowledgment from node v to node u is the only message that node u hears in round j. Thus, every acknowledgment is guaranteed to be received by its recipient at least once.

Therefore, at the end of stage i, there are at most $\Delta/2^{i-1}$ unnumbered assigned neighbors for each DS node. In particular, at the end of stage $1 + \log \Delta$, there are less than 1, i.e., exactly 0 unnumbered assigned neighbors for each DS node. Finally, the local numbering

at each DS node is done using subsequent integers from 1 up to possibly $\Delta+1$, and each node is numbered once (when it transmits successfully to the assigned DS node for the first time). ◀

Below we conclude the analysis of the whole algorithm: the main algorithm, preceded by the two procedures.

▶ **Theorem 16.** *If nodes know n, Δ, γ_2, or their linear upper bounds, then a single bit of advice allows a deterministic distributed Local Broadcast protocol working in $O(\Delta\gamma_2^2 \log^2 n)$ communication rounds. The protocol may need a constant adaptivity (i.e., rounds in which it uses some information to redefine its next schedule), but does not require collision detection or acknowledgments built-in the model.*

Proof. By Lemma 14, after the DS-node Assignment Procedure, each node is assigned to one DS node (in the neighborhood, including itself). By Lemma 15, after the next Numbering Procedure, each node in DS learns about all neighbors assigned to it and give them unique numbers from 2 to $\Delta+1$, reserving number 1 for itself.

Now we analyze the Local Broadcast property of the main algorithm. Suppose, to the contrary, that some node v has not successfully broadcasted to some of its neighbors, u. Let v^*, u^* be the DS nodes assigned to v, u, respectively, by the DS-node Assignment Procedure. Note that both v^*, u^* are in 2-hop neighborhood of u. Therefore, there is a set F_i in the strong selector $\mathcal{F}$ used by the main algorithm such that $v^* \in F_i$ and node of other DS nodes in 2-hop neighborhood of u belongs to F_i. It follows from the property of strong selector $\mathcal{F}$ and the upper bound γ_2 on the number of DS nodes in any 2-hop neighborhood of any node (in particular, node u. It follows than no neighbors of u transmits in rounds $(i-1)\cdot(\Delta+1)+x$, for $x \in \{1, \ldots, \Delta+1\}$, except of those assigned to the DS node v^* (such as node v). Let x^* be the locally unique number of node v. Then, in round $(i-1)\cdot(\Delta+1)+x^*$, also no other node assigned to v^* transmits, by the property of unique numbering of nodes assigned to v^*. Hence, v is the only transmitting node in the neighborhood of u, including u, in that round, and so u receives its message. This contradiction proves that the main algorithm, preceded by the two procedures, guarantees Local Broadcast.

By Lemmas 14 and 15, and by the number of rounds $|\mathcal{F}| \cdot (\Delta+1) \in O(\Delta\gamma_2^2 \log n)$ of the main algorithm, we derive the upper bound $O(\Delta\gamma_2^2 \log^2 n)$ on the total number of rounds. ◀

4 Lower Bound via Hitting Game

In this section, we prove a lower bound for Local Broadcast in the classic RN model with collision detection as defined in Section 2. First, we present a technical tool, of potentially independent interest, to be used later to prove our main lower bound.

4.1 Directed-matching Hitting Game

We prove in this section a lower bound on the number of rounds to finish a directed-matching hitting game between a deterministic algorithm and a randomized adversary on an undirected graph $G=(V,E)$, where V is the set of nodes and E the set of edges. The parameters of the game are the number of nodes n, the maximum degree Δ, and the 2-*local* domination number $\gamma_2 = \gamma_2(G)$ (defined in Section 2). The following additional definitions will be used.

A **directed matching** on a graph $G=(V,E)$ is a set $M = \{(u_1,w_1),(u_2,w_2),(u_3,w_3),\ldots\}$ of ordered pairs of nodes such that the edges $M' = \{\{u_1,w_1\},\{u_2,w_2\},\{u_3,w_3\},\ldots\}$ form a matching in G. We call a subset of nodes $Q \subseteq V$ a **query** on V. Given a query $Q \subseteq V$

and a pair of nodes $u, w \in V$, we say that Q **hits the ordered pair** (u, w) if and only if $\{u, w\} \in E$, $u \in Q$, and for all $v \in Q$ such that $v \neq u$, the following holds: $\{v, w\} \notin E$. Given a set of queries $\mathcal{Q} = \{Q_1, Q_2, \dots\}$ and a directed matching M, we say that $\mathcal{Q}$ **hits the directed matching** M if and only if, for every ordered pair $(u, w) \in M$, there is a query $Q \in \mathcal{Q}$ that hits the ordered pair (u, w).

In the directed-matching hitting game, the adversary and the algorithm compete, with the algorithm trying to hit all the edges of a directed matching chosen by the adversary and the adversary trying to prevent edges of the directed matching from being hit by adding edges under some restrictions. The definition of the game is given in Figure 1.

Directed-matching Hitting Game with parameters γ_2, Δ and n:

- **Game initialization.** Before the game starts, the adversary chooses a directed matching M of size $n/2$ (without loss of generality, we assume n to be even). We label the nodes of this directed matching by $M = \{(u_1, w_1), (u_2, w_2), \dots, (u_{n/2}, w_{n/2})\}$ for reference. We denote $U = \{u_i | i \in [n/2]\}$ and $W = \{w_i | i \in [n/2]\}$. The corresponding set of undirected edges $E_0 = \{\{u_i, w_i\} | i \in [n/2]\}$ is the initial set of edges in the graph. On the other hand, the algorithm chooses a sequence of queries $\langle Q_1, Q_2, \dots \rangle$, hidden from the adversary, where $Q_r \subseteq U$ for any $r = 1, 2, \dots$.
- **Rules of one round of the game.** At the beginning of each round r, the algorithm announces the query $Q_r \setminus H_r$, where H_r contains every node $u_i \in U$ such that the ordered pair (u_i, w_i) has been hit before round r. (Informally, the algorithm is allowed to remove the members of H_r from the query Q_r it prepared in advance.) Then, the adversary may add some edges to the graph, restricted to the requirements:
  - the maximum degree of the resulting graph is at most Δ,
  - the 2-local domination number of the resulting graph is at most γ_2,
  - if an edge $\{u_j, w_i\}$, $i, j \in [n/2]$, is added in round r, then (u_i, w_i) has not been hit before r.
- **Ending conditions.** The game ends in a round τ once (every edge of) the directed matching M has been hit by the queries $\langle Q_1, Q_2, \dots, Q_\tau \rangle$ announced by the algorithm.
- **Objective of each player.** The objective of the algorithm is to minimize the number of rounds τ, whereas the objective of the adversary is to maximize it.

Figure 1 Game definition.

In the above context, we prove the following.

▶ **Theorem 17.** *Consider a directed-matching hitting game with parameters $\gamma_2 > 1$, $\Delta > 1$ and $n \geq 3$. For each algorithm player $\mathcal{P}_A$, there exists an adversary player $\mathcal{P}_B$ such that the number of rounds τ needed to finish the game is $\tau \in \Omega\left(\min\left\{\left(\frac{\min\{\Delta, \gamma_2\}}{\log n}\right)^2, n\right\}\right)$.*

The proof of Theorem 17 (deferred to the full version of this paper) is based on showing a strategy for the adversary for each query of the sequence announced by the algorithm. The strategy is to initially partition the directed matching in *gadgets* (subsets of ordered pairs) at random. Then, add edges to each gadget to make it bipartite-complete. These added edges will guarantee that hits in large queries are very unlikely. Some additional edges are added to each gadget so that a chosen node dominates the gadget. Starting from this initial graph, for each small query the adversary adds edges to prevent hits from nodes that do not appear very frequently in small queries. Other hits are allowed because they are difficult or impossible to prevent. In our analysis we prove that such a strategy fulfills the requirements

of the game, and that within the claimed time lower bound only a fraction of the directed matching can be hit, all with positive probability. Hence, by the probabilistic method, there exists a deterministic strategy that proves the claim.

4.2 Local Broadcast in the RN Model

In this section, we prove a lower bound for Local Broadcast using the lower bound for the directed-matching hitting game proved in Section 4.1. Our lower bound applies to quasi-adaptive protocols, and hence to oblivious protocols as well.

▶ **Theorem 18.** *Consider any deterministic quasi-adaptive protocol $\mathcal{P}$ that solves Local Broadcast in the RN model. Let τ be the number of rounds needed by $\mathcal{P}$ in the worst case. Then, for each $\mathcal{P}$, there exists an adversarial input network with set V of n nodes, 2-local domination number at most γ_2, and maximum degree Δ such that $\tau \in \Omega\left(\min\left\{\left(\frac{\min\{\Delta,\gamma_2\}}{\log n}\right)^2, n\right\}\right)$.*

Proof. For the sake of contradiction, assume there exists a deterministic quasi-adaptive Local Broadcast protocol $\mathcal{P} = \left[b_{v,r}\right]_{v\in V, r\in\tau}$ such that $\tau \in o\left(\min\left\{n, (\min\{\Delta,\gamma_2\}/\log n)^2\right\}\right)$ for all input radio networks with a set V of n nodes, 2-local domination number at most γ_2, and maximum degree Δ. We show in the rest of the proof that then we can use such Local Broadcast protocol (with some modification) as the algorithm player in the game of Section 4.1, and use the network built by the adversary player during the game as the input of the Local Broadcast protocol to reach a contradiction with the lower bound in Theorem 17. The details follow.

Consider a protocol $\mathcal{P}'$ derived from $\mathcal{P}$ by removing half of the schedule. Specifically, for any even partition $V = \{U, W\}$, it is, $\mathcal{P}' = \left[b_{u,r}\right]_{u\in U, r\in\tau}$. (Without loss of generality, we assume that n is even.) Let the **restricted Local Broadcast** problem for a subset of nodes $U \subseteq V$ be the same as the general Local Broadcast, except that it is restricted to messages of nodes in U. That is, each node $u \in U$ holds a message m_u, and the problem is solved once m_u is delivered to every node in $N(u)$ (recall that $N(u)$ is the set of neighbors of u in V). Given the assumption above, protocol $\mathcal{P}'$ solves the *restricted* Local Broadcast problem for U in time $\tau \in o\left(\min\left\{n, (\min\{\Delta,\gamma_2\}/\log n)^2\right\}\right)$. The proof of the latter is a relatively straightforward argument, which we nevertheless include for completeness. Assume, for the sake of contradiction, the opposite. That is, for some input network executing protocol $\mathcal{P}$, the message of some node $u \in U$ is delivered to all nodes in $N(u)$, but if protocol $\mathcal{P}'$ is executed in the same input network, there is a pair of nodes $u \in U$ and $w \in W$, such that $w \in N(u)$ but the message m_u was not received by w. Let r be the round when m_u is delivered to w while running protocol $\mathcal{P}$. If the message was delivered to w it means that no other node in $N(w)$ was transmitting in round r, i.e., $b_{x,r} = false$, for all $x \in N(w)$ such that $x \neq u$. Then, it is also $b_{x,r} = false$, for all $x \in N(w) \setminus W$ such that $x \neq u$. Hence, m_u is also delivered to w in round r (or before) if nodes execute $\mathcal{P}'$, which is a contradiction.

Then, we use $\mathcal{P}'$ as the algorithm player $\mathcal{P}_A$ of the directed-matching hitting game presented in Section 4.1. That is, we use the schedule of transmissions of $\mathcal{P}'$ as the sequence of queries of $\mathcal{P}_A$. Then, following the strategy specified in the proof of Theorem 17, by round τ, the adversary has built a graph $G_\tau = (V, E_\tau)$. According to Theorem 17, with positive probability, G_τ is one of the adversarial graphs that force the game to take $\Omega\left(\min\left\{n, (\min\{\Delta,\gamma_2\}/\log n)^2\right\}\right)$ rounds before hitting the directed matching M embedded in G_τ (in fact, with high probability, as the analysis shows). Notice that due to the restrictions of the adversary in the game (see Figure 1), G_τ has the maximum degree Δ, local 2-domination number at most γ_2, and edges are added only to prevent the hit of matching

ordered-pairs that have not been hit earlier. Thus, edges added during the game would not have prevented hits that happened earlier, had those edges been in the graph from the beginning. Therefore, if G_τ had been used from round 1 of the game, the lower bound on the time to hit M claimed in Theorem 17 still holds.

Consider a radio network with topology graph G_τ. Given our initial assumption that $\mathcal{P}$ solves Local Broadcast on any network in time $o\left(\min\left\{n, (\min\{\Delta, \gamma_2\}/\log n)^2\right\}\right)$, as we showed above, $\mathcal{P}'$ must solve *restricted* Local Broadcast on G_τ also in the same time. However, to solve restricted Local Broadcast on G_τ, the directed matching M must be hit. Thus, the running time of $o\left(\min\left\{n, (\min\{\Delta, \gamma_2\}/\log n)^2\right\}\right)$ is a contradiction with the lower bound in Theorem 17. ◀

5 Conclusions and Open Directions

Most research on RN addressed global problems, often using probabilistic algorithms especially when the network topology is not known. We foray into local problems, in particular Local Broadcast, and the use of deterministic algorithms for them. We demonstrated that in case of oblivious and quasi-adaptive solutions, not much can be improved beyond quite pessimistic lower bounds. We showed on the other hand that a small amount of knowledge and/or a small amount of adaptivity suffice for time complexity improvement.

This work leaves many questions open. What is the inherent complexity of other local tasks? Which global tasks can be performed without using Local Broadcast as a primitive, and thus, possibly have even a lower complexity? Even for Local Broadcast there remain many questions. Our bounds are for the case that all the nodes wish to perform Local Broadcast. Can one derive better algorithms for the case that only some x nodes wish to locally broadcast? Must x be known to help? We provided three points in the tradeoff between adaptivity and knowledge. What is the complete tradeoff?

References

1 Norman Abramson. The aloha system: Another alternative for computer communications. In *Proceedings of the November 17-19, 1970, fall joint computer conference*, pages 281–285, 1970. doi:10.1145/1478462.1478502.

2 Noga Alon, Amotz Bar-Noy, Nathan Linial, and David Peleg. A lower bound for radio broadcast. *Journal of Computer and System Sciences*, 43(2):290–298, 1991. doi:10.1016/0022-0000(91)90015-W.

3 Reuven Bar-Yehuda, Oded Goldreich, and Alon Itai. On the time-complexity of broadcast in multi-hop radio networks: An exponential gap between determinism and randomization. *Journal of Computer and System Sciences*, 45(1):104–126, 1992. doi:10.1016/0022-0000(92)90042-H.

4 Annalisa De Bonis, Leszek Gasieniec, and Ugo Vaccaro. Optimal two-stage algorithms for group testing problems. *SIAM J. Comput.*, 34(5):1253–1270, 2005. doi:10.1137/S0097539703428002.

5 Joan Boyar, Lene M Favrholdt, Christian Kudahl, Kim S Larsen, and Jesper W Mikkelsen. Online algorithms with advice: A survey. *ACM Computing Surveys (CSUR)*, 50(2):1–34, 2017. doi:10.1145/3056461.

6 Gewu Bu, Maria Potop-Butucaru, and Mikaël Rabie. Wireless broadcast with short labels. In *International Conference on Networked Systems*, pages 146–169. Springer, 2020. doi:10.1007/978-3-030-67087-0_10.

7 J. Capetanakis. Tree algorithms for packet broadcast channels. *IEEE Trans. Inf. Theor.*, 25(5):505–515, September 1979. doi:10.1109/TIT.1979.1056093.

8 Mahdi Cheraghchi and Vasileios Nakos. Combinatorial group testing and sparse recovery schemes with near-optimal decoding time. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS'20 2020, Durham, NC, USA, November 16-19, 2020*, pages 1203–1213. IEEE, 2020. doi:10.1109/FOCS46700.2020.00115.

9 Imrich Chlamtac and Shay Kutten. On broadcasting in radio networks-problem analysis and protocol design. *IEEE Transactions on Communications*, 33(12):1240–1246, 1985. doi:10.1109/TCOM.1985.1096245.

10 Bogdan S. Chlebus and Dariusz R. Kowalski. Almost optimal explicit selectors. In Maciej Liskiewicz and Rüdiger Reischuk, editors, *Fundamentals of Computation Theory, 15th International Symposium, FCT 2005, Lübeck, Germany, August 17-20, 2005, Proceedings*, volume 3623 of *Lecture Notes in Computer Science*, pages 270–280. Springer, 2005. doi:10.1007/11537311_24.

11 Israel Cidon and Osnat Mokryn. Propagation and leader election in a multihop broadcast environment. In *Distributed Computing: 12th International Symposium, DISC'98 Andros, Greece, September 24–26, 1998 Proceedings 12*, pages 104–118. Springer, 1998. doi:10.1007/BFB0056477.

12 Andrea EF Clementi, Angelo Monti, and Riccardo Silvestri. Selective families, superimposed codes, and broadcasting on unknown radio networks. In *Proceedings of the twelfth annual ACM-SIAM Symposium on Discrete algorithms (SODA'01)*, pages 709–718, 2001.

13 Andrea E.F. Clementi, Angelo Monti, and Riccardo Silvestri. Distributed broadcast in radio networks of unknown topology. *Theoretical Computer Science*, 302(1):337–364, 2003. doi:10.1016/S0304-3975(02)00851-4.

14 Peter Davies. Optimal message-passing with noisy beeps. In *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing (PODC'23)*, pages 300–309, 2023. doi:10.1145/3583668.3594594.

15 Peter Davies. Uniting general-graph and geometric-based radio networks via independence number parametrization. In *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing (PODC'23)*, pages 290–299, 2023. doi:10.1145/3583668.3594595.

16 Anders Dessmark and Andrzej Pelc. Broadcasting in geometric radio networks. *Journal of Discrete Algorithms*, 5(1):187–201, 2007. doi:10.1016/J.JDA.2006.07.001.

17 Krzysztof Diks, Evangelos Kranakis, Danny Krizanc, and Andrzej Pelc. The impact of information on broadcasting time in linear radio networks. *Theoretical Computer Science*, 287(2):449–471, 2002. doi:10.1016/S0304-3975(01)00256-0.

18 Faith Ellen and Seth Gilbert. Constant-length labelling schemes for faster deterministic radio broadcast. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'20)*, pages 213–222, 2020. doi:10.1145/3350755.3400238.

19 Faith Ellen, Barun Gorain, Avery Miller, and Andrzej Pelc. Constant-length labeling schemes for deterministic radio broadcast. *ACM Transactions on Parallel Computing*, 8(3):1–17, 2021. doi:10.1145/3470633.

20 Paul Erdős, Peter Frankl, and Zoltán Füredi. Families of finite sets in which no set is covered by the union of r others. *Israel J. Math*, 51(1-2):79–89, 1985.

21 Ralph J. Faudree, Zdeněk Ryjáček, and Richard H. Schelp. On local and global independence numbers of a graph. *Discrete Applied Mathematics*, 132(1):79–84, 2003. Stability in Graphs and Related Topics. doi:10.1016/S0166-218X(03)00391-3.

22 Pierre Fraigniaud, David Ilcinkas, and Andrzej Pelc. Communication algorithms with advice. *J. Comput. Syst. Sci.*, 76(3-4):222–232, 2010. doi:10.1016/J.JCSS.2009.07.002.

23 Adam Gańczorz, Tomasz Jurdziński, and Andrzej Pelc. Optimal-length labeling schemes for fast deterministic communication in radio networks. *arXiv preprint arXiv:2410.07382*, 2024.

24 Cyril Gavoille, David Peleg, Stéphane Pérennes, and Ran Raz. Distance labeling in graphs. *Journal of algorithms*, 53(1):85–112, 2004. doi:10.1016/J.JALGOR.2004.05.002.

25 Mohsen Ghaffari and Bernhard Haeupler. Near optimal leader election in multi-hop radio networks. In *Proceedings of the twenty-fourth annual ACM-SIAM Symposium on Discrete algorithms (SODA'13)*, pages 748–766. SIAM, 2013. doi:10.1137/1.9781611973105.54.

26 Seth Gilbert, Nancy A. Lynch, Calvin Newport, and Dominik Pajak. On simple back-off in unreliable radio networks. *Theor. Comput. Sci.*, 806:489–508, 2020. doi:10.1016/J.TCS.2019.08.027.

27 Christian Glacet, Avery Miller, and Andrzej Pelc. Time vs. information tradeoffs for leader election in anonymous trees. *ACM Trans. Algorithms*, 13(3):31:1–31:41, 2017. doi:10.1145/3039870.

28 Barun Gorain and Andrzej Pelc. Finding the size of a radio network with short labels. In *Proceedings of the 19th International Conference on Distributed Computing and Networking*, pages 1–10, 2018. doi:10.1145/3154273.3154298.

29 Barun Gorain and Andrzej Pelc. Finding the size and the diameter of a radio network using short labels. *Theoretical Computer Science*, 864:20–33, 2021. doi:10.1016/J.TCS.2021.02.004.

30 Albert G. Greenberg and Shmuel Winograd. A lower bound on the time needed in the worst case to resolve conflicts deterministically in multiple access channels. *J. ACM*, 32(3):589–596, 1985. doi:10.1145/3828.214125.

31 J.F. Hayes. An adaptive technique for local distribution. *IEEE Communications Magazine*, 19(2), 1981.

32 Michael A. Henning. *Distance Domination in Graphs*, pages 205–250. Springer International Publishing, Cham, 2020. doi:10.1007/978-3-030-51117-3_7.

33 David Ilcinkas, Dariusz R Kowalski, and Andrzej Pelc. Fast radio broadcasting with advice. *Theoretical Computer Science*, 411(14-15):1544–1557, 2010. doi:10.1016/J.TCS.2010.01.004.

34 Michal Katz, Nir A Katz, Amos Korman, and David Peleg. Labeling schemes for flow and connectivity. *SIAM Journal on Computing*, 34(1):23–40, 2004. doi:10.1137/S0097539703433912.

35 William Kautz and Roy Singleton. Nonrandom binary superimposed codes. *IEEE Transactions on Information Theory*, 10(4):363–377, 1964. doi:10.1109/TIT.1964.1053689.

36 Janos Komlós and Albert Greenberg. An asymptotically nonadaptive algorithm for conflict resolution in multiple-access channels. *IEEE Transactions on Information Theory*, 31:302–306, April 1985. doi:10.1109/TIT.1985.1057020.

37 Amos Korman, Shay Kutten, and David Peleg. Proof labeling schemes. In *Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing*, pages 9–18, 2005. doi:10.1145/1073814.1073817.

38 Dariusz R. Kowalski. On selection problem in radio networks. In *Proceedings of the Twenty-fourth Annual ACM Symposium on Principles of Distributed Computing*, PODC '05, pages 158–166, New York, NY, USA, 2005. ACM. doi:10.1145/1073814.1073843.

39 Eyal Kushilevitz and Yishay Mansour. An $\omega(d\backslash\log(n/d))$ lower bound for broadcast in radio networks. *SIAM journal on Computing*, 27(3):702–712, 1998.

40 Nancy Lynch and Calvin Newport. A (truly) local broadcast layer for unreliable radio networks. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing (PODC'15)*, pages 109–118, 2015. doi:10.1145/2767386.2767411.

41 Robert Metcalfe and David R. Boggs. Ethernet: Distributed packet switching for local computer networks. *Commun. ACM*, 19(7):395–404, 1976. doi:10.1145/360248.360253.

42 David Peleg. *Distributed computing: a locality-sensitive approach*. SIAM, Philadelphia, PA, USA, 2000.

43 David Peleg. Time-efficient broadcasting in radio networks: A review. In Tomasz Janowski and Hrushikesha Mohanty, editors, *Distributed Computing and Internet Technology, 4th International Conference, ICDCIT 2007, Bangalore, India, December 17-20, Proceedings*, volume 4882 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2007. doi:10.1007/978-3-540-77115-9_1.

44 B S. Tsybakov and V A. Mikhailov. Free synchronous packet access in broadcast channel with feedback. *Probl Inf Transm*, 14(4), October 1978.

45 Mikkel Thorup and Uri Zwick. Approximate distance oracles. *Journal of the ACM (JACM)*, 52(1):1–24, 2005. doi:10.1145/1044731.1044732.

Anonymous Self-Stabilising Localisation via Spatial Population Protocols

Leszek Gąsieniec
Department of Computer Science, University of Liverpool, UK

Łukasz Kuszner
Institute of Informatics, University of Gdańsk, Poland

Ehsan Latif
AI4STEM Education Center, University of Georgia, Athens, GA, USA

Ramviyas Parasuraman
School of Computing, University of Georgia, Athens, GA, USA

Paul Spirakis
Department of Computer Science, University of Liverpool, UK

Grzegorz Stachowiak
Institute of Computer Science, University of Wrocław, Poland

Abstract

In the *distributed localisation problem* (DLP), n anonymous robots (agents) $A_0, \ldots, A_{n-1}$ are located at arbitrary points $p_0, \ldots, p_{n-1} \in S$, where S is a Euclidean space. Initially, each agent A_i operates within its own coordinate system in S, which may be inconsistent with those of other agents. The primary goal in DLP is for agents to reach a consensus on a unified (jointly agreed) coordinate system, in which all agents receive unique *labels* (coordinates) that accurately reflect the relative distances between all points $p_0, \ldots, p_{n-1}$ in S. Extensive research on DLP has primarily focus on the feasibility and complexity of achieving consensus when agents have limited access to inter-agent distances, often due to missing or imprecise data. In contrast, this paper proposes a minimalist, computationally efficient distributed computing model where agents can query any pairwise relative positions, if needed. Specifically, we introduce a novel variant of population protocols, referred to as the *spatial population protocols* model. In this variant each agent can memorise one or a fixed number of coordinates, and when agents A_i and A_j interact, they can not only exchange their current knowledge but also either determine the distance d_{ij} between them in S (distance query model) or obtain the vector $\overrightarrow{v_{ij}}$ spanning points p_i and p_j (vector query model).

We propose and analyse several distributed localisation protocols, including:

1. *Leader-based localisation protocol with distance queries* We propose and analyse two leader-based localisation protocols that stabilise silently in $o(n)$ time. These protocols leverage an efficient solution to the novel concept of *multi-contact epidemic*, a natural generalisation of the core communication tool in population protocols, known as the *one-way epidemic*.
2. *Self-stabilising leader localisation protocol with distance queries* We show how to effectively utilise a leader election mechanism within the leader-based localisation protocol to get a DLP protocol that self-stabilises silently in time $O(n(\log n/n)^{1/(k+1)} \log n)$ in k-dimensions.
3. *Self-stabilising localisation protocol with vector queries* We propose and analyse an optimally fast DLP protocol which self-stabilises silently in $O(\log n)$ time.

2012 ACM Subject Classification Theory of computation → Distributed computing models

Keywords and phrases Population Protocols, Distributed Localisation, Spacial Queries, Self-Stabilisation

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.35

Related Version *Previous Version*: https://arxiv.org/abs/2411.08434
Previous Version: https://doi.org/10.4230/LIPIcs.SAND.2025.19

© Leszek Gąsieniec, Łukasz Kuszner, Ehsan Latif, Ramviyas Parasuraman, Paul Spirakis, and Grzegorz Stachowiak;
licensed under Creative Commons License CC-BY 4.0

Funding *Paul Spirakis*: Supported by the EPSRC grant EP/P02002X/1.
Grzegorz Stachowiak: Supported by the NCN grant 2020/39/B/ST6/03288.

1 Introduction

Location services are crucial for modern computing paradigms, such as pervasive computing and sensor networks. While manual configuration and GPS can determine node locations, these methods are impractical in large-scale or obstructed environments. Recent approaches use network localisation, where beacon nodes with known positions enable other nodes to estimate their locations via distance measurements. Key challenges remain, including determining conditions for unique localisability, computational complexity, and deployment considerations. In the *distributed localisation problem* (DLP), n anonymous robots (agents) $A_0, \ldots, A_{n-1}$ located at arbitrary points $p_0, \ldots, p_{n-1} \in S$, where S is a Euclidean space. Initially, each agent A_i operates within its own coordinate system in S, which may be inconsistent with those of other agents. The primary goal in DLP is for agents to position themselves by adopting labels (coordinates) in a unified (jointly agreed) coordinate system that accurately reflects the relative distances between all points locations $p_0, \ldots, p_{n-1}$ in S.

A network's unique localisability depends on its graph's combinatorial properties and the number of anchors (agents with known locations). Graph rigidity theory [22, 28, 31, 36] states that a planar network is uniquely localisable with at least three anchors and a globally rigid graph. Global rigidity is rare in non-dense networks, though parts may be rigid, leaving some node positions indeterminate [22]. The graph embedding problem, determining if a weighted graph can be embedded in the plane to match edge weights, is strongly NP-hard [39], even for globally rigid graphs [22]. In sensor networks, modelled as unit disk graphs (adjacent nodes within distance r), the unit disk graph reconstruction problem, ensuring adjacent nodes are within r and non-adjacent nodes are farther apart, is also NP-hard [8]. No efficient algorithm exists for localisation unless P = NP, even for unique reconstructions [8]. Despite distributed computing models like population protocols [24], localisation remains unstudied in these frameworks.

Distributed localisation is also vital for *robotic systems*, enabling autonomous spatial positioning for navigation, mapping, and multi-robot coordination [46]. It encompasses centralised systems, where a central server computes positions, offering accuracy but limited scalability [34], and distributed systems, where robots independently or collaboratively localise, enhancing resilience but increasing complexity [32, 33]. Leader-based methods use designated robots as reference points [23], while leaderless approaches, leveraging probabilistic [43], geometric [35], or graph-based models [32], excel in decentralised, fault-tolerant settings [41]. Tools like Kalman filters [37], particle filters [34], and graph rigidity theory [42] improve accuracy and efficiency.

A strongly related variant of the localisation problem, known as the *n-point location problem*, has been studied in more centralised setting [17, 18]. This problem was motivated by and is closely related to the computation of relative positions of markers on a DNA string [29, 30, 38]. In the *n-point location problem*, the objective is to design one or more rounds of pairwise distance queries between points to determine the relative distances among all points. In [17], a one-round deterministic strategy is presented that utilises $\frac{8n}{5}$ distance queries, along with a proof that any one-round strategy requires at least $\frac{4n}{3}$ distance queries. The same work also introduces a simple two-round deterministic strategy that uses $\frac{3n}{2}$ queries. An alternative two-round randomised approach, which requires only $n + O\left(\frac{n}{\log n}\right)$ queries and solves the *n-point location problem* with high probability, is described in [18].

1.1 Spatial population protocols

This paper presents a minimalist, efficient distributed computing model where randomly paired agents can query their relative positions upon meeting. Our focus is on achieving *anonymity* while maintaining high time efficiency and minimal use of network resources, including limited local storage (agent state space) and communication. To meet these goals, we introduce a new variant of population protocols, referred to as the *spatial population protocols* model.

The *population protocol model* originates from the seminal work by Angluin et al. [6]. This model provides tools for the formal analysis of *pairwise interactions* between indistinguishable entities known as *agents*, which have limited storage, communication, and computational capabilities. In a nutshell, *population protocols* comprise three major components: (1) a *population of agents* $A_0, \ldots, A_{n-1}$, which are simple computational entities with limited states that interact in pairs; (2) *interaction rules* forming a *transition function*, which specify how agents update their states based on pairwise encounters; and (3) a (possibly random) *scheduler of interactions*, which determines the sequence of agent interactions, guiding the system toward a desired output or consensus. In *self-stabilising* protocols, the *initial configuration* of agents' states is arbitrary. By contrast, non-self-stabilising protocols start with a predefined configuration encoding the input of the considered problem. A protocol concludes when it *stabilises* in a *final configuration* of states representing the solution of the considered problem. In the *probabilistic variant* of population protocols, which is also used here, the *random scheduler* selects an ordered pair of agents at each step, designated as the *initiator* and the *responder*, uniformly at random from the entire population. The asymmetry in this pair introduces a valuable source of random bits, which is utilised by population protocols. In this probabilistic setting, besides *efficient state utilisation*, *time complexity* is also a primary concern. It is often measured by the number of interactions, $\mathcal{I}$, required for the protocol to stabilise in a final configuration. More recently, the focus has shifted to *parallel stabilisation time* (or simply *time*), defined as $\mathcal{I}/n$, where n is the population size. This measure captures the parallelism of independent, simultaneous interactions, which is leveraged in *efficient population protocols* that stabilise in time $O(\text{poly}\log n)$. All protocols introduced in this paper are *stable* (consistently correct), *stabilise silently* (agent states cease to change post-stabilisation), and guarantee stabilisation time with high probability (whp), defined as $1 - n^{-\eta}$, for a constant $\eta > 0$. Furthermore, our primary contribution, the localisation protocols detailed in Sections 3 and 4, exhibits self-stabilisation.

Among the primary tools in our localisation protocols, we emphasise the novel concept of *multi-contact epidemic*, see Section 2, a natural extension of the fundamental communication mechanism in population protocols, known as the *one-way epidemic*. Another key tool is *leader election*, a cornerstone of distributed computing that tackles the essential challenges of symmetry breaking, synchronisation, and coordination. In population protocols, the presence of a leader facilitates a more efficient computational framework [7]. However, achieving leader election within this model presents significant difficulties. Foundational results [16, 19] demonstrate that it cannot be solved in sublinear time if agents are restricted to a fixed number of states [20]. Further, Alistarh and Gelashvili [3] introduced a protocol stabilising in $O(\log^3 n)$ time with $O(\log^3 n)$ states. Later, Alistarh *et al.* [1] identified trade-offs between state use and stabilisation time, distinguishing *slowly* ($o(\log\log n)$ states) and *rapidly stabilising* ($O(\log n)$ states) protocols. Subsequent work achieved $O(\log^2 n)$ time whp and in expectation with $O(\log^2 n)$ states [13], later reduced to $O(\log n)$ states using synthetic coins [2, 12]. Recent research by Gąsieniec and Stachowiak reduced state usage to $O(\log\log n)$ while retaining $O(\log^2 n)$ time whp [26]. The expected time of leader election

was further optimised to $O(\log n \log\log n)$ by Gąsieniec *et al.* in [27] and to the optimal time $O(\log n)$ by Berenbrink *et al.* in [11]. In contrast, self-stabilising leader election protocols present unique computational challenges. Notably, it was shown by Cai *et al.* in [15] that such protocols require at least n states, in addition to knowledge of the exact value of n. Alternatively, if loose-stabilisation is allowed (where a leader remains in place for a long but finite time before re-election) an upper bound on n may suffice, see work of Sudo *et al.* [40]. In [14, 21], it was established that any silently self-stabilising leader election protocol has an expected time complexity of at least $\Omega(n)$. Furthermore, Burman *et al.* in [14] present silently self-stabilising protocols with expected time $O(n)$ and with high probability (whp) time $O(n \log n)$, both using $n+\Omega(n)$ states. More recently, the space complexity of $O(n \log n)$-time leader election self-stabilising whp protocol has been improved to $n + O(\log^2 n)$ in [10] and to $n + O(\log n)$ in [25]. In this work, however, we employ a relatively straightforward leader election mechanism where each agent collects $\log n$ random bits, and a leader is determined by a complete set of $\log n$ 1s. This simple protocol ensures the election of a unique leader with constant probability. When repeated $O(\log n)$ times, this procedure guarantees unique leader election whp. For details see Section 3.

Spatial embedding with geometric queries

While population protocols offer an elegant and resilient framework for randomised distributed computation, with the exception of work on *graphical models* [9, 44, 5, 4, 45], they typically lack topological (in particular, geometric) embedding. To address this limitation, we propose a *spatial variant* of probabilistic population protocols that extends the state space and transition function to accommodate unspecified agent locations $p_0, \ldots, p_{n-1}$ and two fundamental *geometric queries*.

More specifically, in the spatial model each agent can memorise a fixed number of hypothetical or agreed coordinates, and during an interaction of two agents A_i and A_j, in addition to exchange of their current knowledge, the agents can determine:
(1) the distance d_{ij} separating them in S, in *distance query model*, and
(2) vector $\overrightarrow{v_{ij}}$ spanning points p_i and p_j, in *vector query model*.

As this is the initial work on spatial population protocols, we assume the random scheduler remains unchanged, with pairs of agents selected uniformly at random for interactions. In future work, it would be worth considering models where the probability of agent interactions vary (e.g., in relation to pairwise relative distances) as well as other geometric queries.

1.2 Importance and structure of presentation

As previously discussed, the localisation problem is fundamental, with its many variants extensively studied over time. This work effectively integrates research in (distributed) localisation with rigidity theory of random geometric graphs, leveraging the increasingly popular population protocol model of computation. Surprisingly, little research has explored population protocols incorporating spatial properties, particularly those supporting geometric queries. The most relevant prior work focuses on efficient majority and leader election protocols for general toroidal grids [4] and self-stabilising leader election in rings [44, 45].

In this study, we assume agents are distributed in a k-dimensional space, where each interaction, viewed as a pairwise geometric query, results in an exchange of states (knowledge held by agents) augmented by information about the relative positions of the interacting agents. We present the following results.

In Section 2, we propose and analyse two leader-based localisation protocols in the model with distance queries. The first protocol (Algorithms 1) stabilises silently in time $O(n(\log n/n)^{1/(k+1)})$ whp (Theorem 4), in k-dimensional space. For $k = 1$, we present a faster protocol with the stabilisation time $O((n \log n)^{1/3})$ whp (Theorem 9). These two protocols leverage an efficient solution to the novel concept of *multi-contact epidemic* (Section 2.1), a natural generalisation of the core communication tool in population protocols, known as the *one-way epidemic*. In Section 3, we show how to effectively utilise leader election within the leader-based localisation protocol (Algorithm 1) in the model with distance queries. We propose and analyse a DLP protocol that self-stabilises silently in time $O(n(\log n/n)^{1/(k+1)} \log n)$ whp (Theorem 13), in k-dimensional space. Finally, in Section 4, we present surprisingly simple and optimally fast DLP protocol which self-stabilises silently in $O(\log n)$ time whp (Theorem 15), in the model with vector queries and fixed k.

For simplicity we assume that all geometric calculations are done via real arithmetic. Alternatively, one may assume that the actual positions of the agents are on vertices of a grid with integer coordinates, and, when determining the position of an agent, the agent can round the position up to the closest grid point.

2 Leader-based localisation in distance query model

In this section, we discuss two localisation protocols with predefined input stabilising in $o(n)$ time, i.e., after this time labels of all agents become stable. These protocols are non-self-stabilising. We assume that one of the agents starts as the *leader* of the population. The agents' point positions $p_0, \ldots, p_{n-1}$ are distributed in a k-dimensional Euclidean space S. It is assumed that any $k+1$ agents' positions span the entire space. For example, in two-dimensional space, this assumption guarantees that no three positions are collinear. Although our algorithms can be adapted to handle an arbitrary distribution of agents' positions, the time guarantees of such adaptations would be weaker. The state of an agent can accommodate a fixed (related to k) number of agent positions and distances.

We adopt a symmetric model of communication, which means that when agents A_u and A_v interact, they both gain access to each other's states as well as the distance d_{uv}. The early transitions assigned to the leader are distinct from those of the other agents, as the leader also serves as the initiator of the entire process. Initially, the state of each agent A_u stores a label x_u (representing a hypothetical position in S) and its colour $C[A_u]$. We assume that at the beginning, only the leader is coloured `green`, and each non-leader's colour is set to `blue`. Finally, the leader's label (position in S) is set at the origin of the coordinate system, i.e., this label is used as the anchor in the localisation process. In due course, each agent eventually becomes `green`, adopting a label (position) consistent with all other `green` agents upon doing so. Before presenting the localisation protocol, we introduce a novel concept of *multi-contact epidemic*, which is pivotal to our proposed solution.

2.1 Multi-contact epidemic

We introduce and analyse the process of κ-*contact epidemic* (a multi-contact epidemic with a fixed parameter κ), a natural generalisation of epidemic dynamics in population protocols. In this process, the population initially contains at least κ `green` agents, while the remaining agents are `blue`. A `blue` agent turns `green` after interacting with κ distinct `green` agents. We show that the time complexity of this process is $O(n^{1-1/\kappa} \log n)$, for any fixed integer κ.

We start with two key lemmas that outline the progression of nominating `green` agents in the multi-contact epidemic.

▶ **Lemma 1.** *The time needed to transition from a configuration with m `green` agents to a configuration with $2m$ `green` agents for $m < n^{1/\kappa}\log n$ is bounded by*

$$(c+\kappa)\frac{n^{1-1/\kappa}(\log n)^{1/\kappa}}{\sqrt{m-\kappa+1}}$$

with high probability (whp), for some constant $c > 0$.

Proof. Assume that the number of `green` agents is exactly m. Consider $\kappa - 1$ successive periods of length $n^{1-1/\kappa}(\log n)^{1/\kappa}/\sqrt{m-\kappa+1}$ and an additional period of length $cn^{1-1/\kappa}(\log n)^{1/\kappa}/\sqrt{m-\kappa+1}$. We show that after all these periods, we obtain at least m new green agents whp.

We prove by induction that after the first i periods there are at least n_i agents that had at least i contacts (interactions with distinct `green` agents) for $i = 0, 1, 2, \ldots, \kappa - 1$, where $n_0 = n - m$ and for $i > 0$,

$$n_i = m(m-1)(m-2)\cdots(m-i+1)\cdot n\left(\frac{\log n}{n}\right)^{i/\kappa}(m-\kappa+1)^{-i/2}.$$

Now we prove the inductive step. Assume that after initial $i-1$ periods there are at least n_{i-1} agents with at least $i-1$ contacts. Let X_t be a random variable that equals 1 if, at time t, an agent with $i-1$ contacts experiences a new contact (with its i-th `green` agent), and 0 otherwise. If in time t less than n_i agents had i contacts

$$\Pr(X_t = 1) > 2(m-i+1)\frac{n_{i-1}-n_i}{n(n-1)} > 1.5(m-i+1)\frac{n_{i-1}}{n^2}.$$

After the i-th period of length $n^{1-1/\kappa}(\log n)^{1/\kappa}/\sqrt{m-\kappa+1}$ the expected value of $EX = \sum_t EX_t$ is at least

$$1.5(m-i+1)\frac{n_{i-1}}{n^2}\cdot n\cdot\frac{n^{1-1/\kappa}(\log n)^{1/\kappa}}{\sqrt{m-\kappa+1}} > 1.4n_i.$$

By Lemma 5 this number is at least n_i whp. Thus also the number of agents that had at least i contacts is at least n_i whp.

After $\kappa - 1$ periods of length $n^{1-1/\kappa}(\log n)^{1/\kappa}/\sqrt{m-\kappa+1}$ there are at least $n_{\kappa-1}$ agents that had at least $\kappa - 1$ contacts. We show that after an additional period of length $cn^{1-1/\kappa}(\log n)^{1/\kappa}/\sqrt{m-\kappa+1}$ we will have at least m new green agents whp. Let X_t be a random variable that equals 1 if, at time t, an agent with $\kappa-1$ contacts experiences a new contact (with its κ-th `green` agent), and 0 otherwise. Note that each time $X_t = 1$, a new `green` agent is produced. As long as less than m agents became green,

$$\Pr(X_t = 1) > 2(m-\kappa+1)\frac{n_{\kappa-1}-m}{n(n-1)} > \frac{n_{\kappa-1}}{n^2}.$$

After one extra period of length $cn^{1-1/\kappa}(\log n)^{1/\kappa}/\sqrt{m-\kappa+1}$ the expected value $EX = \sum_t EX_t$ is at least $cm(m-1)\cdots(m-\kappa+1)\cdot m^{-\kappa/2}\log n > cm\log n$. By Lemma 5 this number is at least m whp. Thus, also the number of newly generated `green` agents is at least m whp. ◀

▶ **Lemma 2.** *Starting with at least $n^{1/\kappa}\log n$ `green` agents guarantees recolouring all n agents to `green` in time $O(n^{1-1/\kappa})$ whp.*

Proof. If there are altogether $n^{1/\kappa}\log n$ `green` agents, then for any `blue` agent one can define a random variable X_t equal 1 if during interaction t this agent interacts with a new `green` agent, and 0 otherwise. We have $\Pr(X_t = 1) > 1.5n^{1/\kappa}\log n/n$. In time $c\kappa n^{1-1/\kappa}$ the value $EX = \sum EX_t > 1.5c\kappa\log n$. By Lemma 5 for c big enough $X \geq \kappa$ whp, i.e., each agent becomes `green` whp. ◀

▶ **Theorem 3.** *The stabilisation time of κ-contact epidemic is $O(n^{1-1/\kappa}\log^{1/\kappa} n)$ whp.*

Proof. The execution time of κ-contact epidemic is the sum of the times to increase the number of `green` agents from $m = \kappa$ to $m = n$, and can be calculated using Lemmas 1 and 2

$$T = O(n^{1-1/\kappa}) + \sum_{m=\kappa,2\kappa,4\kappa,8\kappa,\ldots,n^{1/\kappa}\log n} (c+\kappa)\frac{n^{1-1/\kappa}(\log n)^{1/\kappa}}{\sqrt{m-\kappa+1}} = O(n^{1-1/\kappa}\log^{1/\kappa} n),$$

since we assumed that κ is fixed. ◀

2.2 Localisation via multi-contact epidemic in k-dimensions

The localisation protocol consists of two parts (see Algorithm 1). In Part 1: *nomination process* (lines 9-15), the leader nominates and labels k agents, s.t., their labels (coordinates) span k-dimensional space in a common coordinate system. Their labels stabilise and agents become `green`. In Part 2: *multilateration process* (lines 16-22), the labels of all remaining (`blue`) agents become stable (`green`). And this happens when each remaining (`blue`) agent manages to interact with $k+1$ different `green` agents. This sequence of interactions enables each agent to position itself within a mutually agreed-upon coordinate system. Part 2 is based on a direct application of κ-contact epidemic, for $\kappa = k+1$, as discussed in Section 2.1.

In Part 1, the positioning (labelling) of the first k `green` agents requires nomination and approval from the leader, who maintains its private list of nominated `green` agents. More precisely, after an aspiring to become `green` agent A_v manages to interact with all $i < k$ currently existing `green` agents. To become `green` A_v must meet the leader and verify its list of `green` contacts to get nomination. Upon successful verification, the leader updates its list of `green` agents, and the new `green` agent A_v is ready to calculate its projection onto the subspace spanned by its i `green` predecessors and the leader, as well as its Euclidean distance from this subspace. Namely, the first i coordinates of A_v's label are determined by this projection, and the $(i+1)$-th coordinate (in the newly formed dimension) is equal to its distance from the aforementioned subspace. In Part 2, when positioning the remaining agents, we use the fact that interactions with $k+1$ `green` agents allow for the unambiguous determination of an agent's position in the commonly agreed coordinate system.

▶ **Theorem 4.** *Algorithm 1 stabilises silently in time $O(n(\log n/n)^{1/(k+1)})$ whp.*

Proof. As mentioned earlier, Algorithm 1 operates in two parts. In Part 1, the protocol stabilises the labels of the leader and k extra agents, creating an initial set of $k+1$ `green` agents in time $O(n(\log n/n)^{1/k})$ whp (Lemma 6). The second part stabilises labels of all remaining agents via $(k+1)$-contact epidemic in time $O(n(\log n/n)^{1/(k+1)})$ (Theorem 3). ◀

First we formulate the following lemma.

▶ **Lemma 5.** *Consider $X = X_1 + X_2 + \cdots + X_n$ of n independent $0-1$ random variables and any $\delta > 0$. If the expected value $EX \geq c\log n$, for $c > 0$ large enough, then $|X - EX| < \delta EX$ holds whp.*

Algorithm 1 Positioning in k dimensions.

```
/* Code executed by A_u exchanging data from A_v during interaction */
local state    : x – position, C – colour, L – list of green agents' (position,distance)
received data: L(A_v), C(A_v), x(A_v), d_uv
1  begin
   /* Initial values                                                  */
2    L(A_u) ← empty list
3    if u is the leader then
4      C(A_u) ← green; x(A_u) ← (0, ..., 0)
5    else
6      C(A_u) ← blue
7    end
8    while not all agents are positioned do
9      if A_u is the leader ∧ C(A_v) = blue ∧ |L(A_u)| = |L(A_v) \ (x(A_u), d_uv)| < k
         then
10       append (L(A_u), (x(A_v), d_uv))
11     end
12     if A_v is the leader ∧ C(A_u) = blue ∧ |L(A_u) \ (x(A_v), d_vu)| = |L(A_v)| < k
         then
13       C(A_u) ← green
14       x(A_u) ← a position consistent with L(A_u) and (x(A_v), d_uv)
15     end
16     if C(A_u) is blue ∧ C(A_v) = green ∧ (A_v, d_uv) ∉ L(A_u) then
17       append (L(A_u), (x(A_v), d_uv))
18       if |L(A_u)| = k + 1 then
19         C(A_u) ← green
20         x(A_u) ← the exact position calculated using L(A_u)
21       end
22     end
23   end
24 end
```

Proof. The following equality holds

$$\Pr(|X - EX| < \delta EX) = \Pr(X > (1+\delta)EX) + \Pr(X < (1-\delta)EX)$$

By Chernoff inequalities, for c large enough and any parameter η, we get

$$\Pr(X > (1+\delta)EX) < e^{-EX\delta^2/(2+\delta)} < e^{-c\delta \log n} < n^{-\eta}/2$$

and

$$\Pr(X < (1-\delta)EX) < e^{-EX\delta^2/2} < e^{-c\delta^2 \log n/2} < n^{-\eta}/2.$$

Thus, $\Pr(|X - EX| < \delta EX) < n^{-\eta}$. ◀

Now we formulate another lemma determining the time needed to position i-th agent during the process of positioning the first k green agents.

▶ **Lemma 6.** *Algorithm 1 (Part 1), the i-th* `green` *agent is positioned in parallel time* $O(n^{1-1/i}(\log n)^{1/i})$ *whp.*

Proof. Consider $i-1$ successive periods of length $n^{1-1/i}(\log n)^{1/i}$ and an additional period of length $cn^{1-1/i}(\log n)^{1/i}$. We show that after all these periods, whp a new green agent is positioned.

We now prove by induction that after the first j periods there are at least n_j agents that had at least j interactions with green agents for $j = 0, 1, 2, \ldots, i-1$, where $n_0 = n-1$ and for $j > 0$, $n_j = n\left(\frac{\log n}{n}\right)^{j/i}$.

We start with the inductive step. Assume that after initial $j-1$ periods there are at least n_{j-1} agents with at least $j-1$ *contacts*, i.e., interactions with at least $j-1$ `green` agents. Let X_t be a random variable that equals 1 if, at time t, an agent with $j-1$ contacts has a new contact with its j-th `green` agent, and 0 otherwise. If in time t less than n_j agents had j contacts then

$$\Pr(X_t = 1) > 2\frac{n_{j-1} - n_j}{n(n-1)} > 1.5\frac{n_{j-1}}{n^2}.$$

After the j-th period of length $n^{1-1/i}(\log n)^{1/i}$ the expected value of $EX = \sum_t EX_t$ is at least

$$1.5\frac{n_{j-1}}{n^2} \cdot n \cdot n^{1-1/i}(\log n)^{1/i} > 1.4n_j.$$

By Lemma 5 this number is at least n_j whp. Thus also the number of agents that had interactions with at least j green agents is at least n_j whp.

Furthermore, after $i-1$ periods there are at least $n_{i-1} = (n/\log n)^{1/i}\log n$ agents that experienced $i-1$ contacts. Consider an extra period of length $cn^{1-1/i}(\log n)^{1/i}$. Let X_t be a random variable that equals 1 if, at time t, an agent with $i-1$ contacts has a contact with the leader, and 0 otherwise. If in time t none of these agents had interaction with the leader yet, we get

$$\Pr(X_t = 1) > 2\frac{n_{i-1}}{n^2}.$$

After an extra period of length $n^{1-1/i}(\log n)^{1/i}$ the expected value of $EX = \sum_t EX_t$ is at least

$$2\frac{n_{j-1}}{n^2} \cdot n \cdot cn^{1-1/i}(\log n)^{1/i} = 2c\log n.$$

And by Lemma 5 this number is at least 1 whp. ◀

2.3 Faster positioning algorithm on the line

In this section, we demonstrate how to enhance performance of Algorithm 1 on the line, i.e., within a linear space S. While we focus on one dimension, our observations can be utilised also in higher dimensions. In what follows, we propose an alternative localisation protocol which positions agents not only by using 2-contact epidemic but also utilising interactions between agents with a single `green` contact on their list, coloured as `greenish`. In the new algorithm we take advantage of the fact that there are only two types of interactions between `greenish` agents not leading to positioning of both of them. The first type refers to interactions between agents that became `greenish` via contacting the same `green` agent. In the second, each of the two interacting `greenish` agents is at the same distance and on the same side of their unique `green` contact, rendering positioning impossible. We show that these types of interactions do not contribute significantly to the total time of the stabilisation.

▶ **Lemma 7.** *The time needed to increase the number of* `green` *agents from m to $2m$, for $m \in [2, n^{0.9}]$, is $O((n \log n/m)^{1/3})$ whp.*

Proof. First we consider an initial period I_1 of length $(n \log n/m)^{1/3}$. The average number of `greenish` agents produced by m `green` agents in time $(n \log n/m)^{1/3}$ is $(m^2 n \log n)^{1/3}$. By Lemma 5 this number is at least $0.9(m^2 n \log n)^{1/3}$ whp. For any given `green` agent A_u, one can define a subset $\mathcal{S}_u$ of `greenish` agents originating from contact with agents other than A_u. The cardinality of S_u is on average at least $0.9(m-1)(n \log n/m)^{1/3}$. By Lemma 5 this number is greater than $0.8(m-1)(n \log n/m)^{1/3}$ whp. For a given agent A_v, which turned `greenish` after contacting A_u, there are fewer than $2m$ `greenish` agents in $\mathcal{S}_u$ with whom no interaction leads to the positioning of A_v, unless the number of `green` agents exceeds $2m$ (they are the second kind of non-positioning agents). Let us consider any set Z of at most $2m$ agents located at points that are translations of the positions of `green` agents by a fixed vector. The expected number of `greenish` agents belonging to Z is at most $n \cdot (n \log n/m)^{1/3} \cdot (2m)^2/n^2 = 4(m^2 n \log n)^{1/3} m/n$. So by Chernoff bound this number is at most $0.1(m^2 n \log n)^{-1/3}$ whp.

Thus for a given `greenish` agent A_v, the number of other `greenish` agents with whom interactions position A_v is whp at least

$$0.8(m-1)\left(\frac{n \log n}{m}\right)^{1/3} - 0.1(m^2 n \log n)^{1/3} > 0.6(m^2 n \log n)^{1/3}.$$

Let I_2 be a time interval of length $c(n \log n/m)^{1/3}$ that follows immediately after I_1. The probability that an interaction t between two `greenish` agents is the one that positions them and makes them `green` is at least $0.6(m^2 n^{-2} \log n)^{2/3}$. An average number of interactions that position pairs of `greenish` agents in period I_2 is $0.6cm \log n$. By Lemma 5 this yields at least m new `green` agents whp. ◀

▶ **Lemma 8.** *The time in which the number of green agents increases from $n^{0.9}$ to n is $O(n^{0.1} \log n)$.*

Proof. Consider all interaction of agent A_u during period $cn^{0.1} \log n$. The probability of having at most one interaction with a green agent for A_u is

$$\left(1 - \frac{2n^{0.9}}{n^2}\right)^{cn^{1.1} \log n} + cn^{1.1}(\log n)0.9n^2 \left(1 - \frac{2n^{0.9}}{n^2}\right)^{cn^{1.1} \log n - 1} < \\ < c(\log n)e^{c \log n} < e^{2c \log n} = n^{2c \ln 2}. \quad (1)$$

So, there is $c > 0$, s.t., all agents have at least two interactions with green agents during the considered period $cn^{0.1} \log n$ whp. ◀

▶ **Theorem 9.** *The stabilisation time of the improved algorithm is $O((n \log n)^{1/3})$ whp.*

Proof. The stabilisation time can be bounded by the sum of the chunks of time needed to increase the number of `green` agents m from 2 to n utilising Lemmas 7 and 8

$$T = O(n^{0.1} \log n) + \sum_{m=2,4,8,\ldots,n^{0.9}} O((n \log n/m)^{1/3}) = O((n \log n)^{1/3}).$$ ◀

3 Self-stabilising localisation in k dimensions

We begin with a brief description of the solution. The proposed self-stabilising protocol in k-dimensions is based on iterative *rounds*, with each round utilising three mechanisms: *leader election* incorporated into *leader-based localisation* (the first stage of a round), and a *buffering mechanism* (the second stage of a round) adopted from [25] (Section 5.2).

During leader election, each agent draws $\log n$ random bits, and if none of these bits is 0, the agent proceeds to the actual localisation protocol as a leader. Note that after all agents complete drawing random bits, which takes $O(\log n)$ time, a unique leader is elected with constant probability, see Lemma 10. Regardless of whether a unique leader is elected, the leader-based localisation protocol continues (possibly indefinitely) unless an *anomaly* is detected. Two types of anomalies may arise. The first, *label inconsistency*, occurs only between two `green` agents. This kind of anomaly occurs when either the initial configuration has conflicting labels of `green` agents or multiple leaders were elected during the current round. When all agents are `green` this type of anomaly is detected in time $O(k \log n)$, see Lemma 11. The second anomaly occurs when any non-`green` agent's counter reaches its deadline $O(n(\log n/n)^{1/(k+1)})$, i.e., the expected stabilisation time of Algorithm 1, indicating that the localisation process is not completed on time. Upon detecting any anomaly, a reset signal is initiated and propagated via a simple epidemic in $O(\log n)$ time. This, together with the (collection and) buffering mechanism adopted from [25] (Section 5.2), ensures sufficient time, namely $O(\log n)$, to reset states of all agents, and prepare them for the next round.

Finally, the time of each round is dominated by the expected time of localisation stabilisation $O(n(\log n/n)^{1/(k+1)})$. Also, the probability of stabilisation in a round is constant, primarily driven by the constant probability associated with successful leader election. Thus, after $O(\log n)$ rounds and a total time of $O(n(\log n/n)^{1/(k+1)} \log n)$, the localisation protocol self-stabilises with high probability (whp), see Theorem 13.

Our result demonstrates that self-stabilisation is not only feasible but also efficient, i.e., comparable to Algorithm 1 presented in Section 2. Moreover, although our self-stabilising localisation protocol relies on leader election, it only requires knowledge of $\log n$, whereas self-stabilising leader election requires precise knowledge of n, see [15]. This is not a contradiction because we only need leader election to succeed with constant probability, which together with efficient anomaly detection, discussed in Section 3.3, and lightly modified buffering mechanism from [25], discussed in Section 3.4, allows us to restart leader election and in turn the localisation protocol, when necessary.

3.1 Memory utilisation

We begin by analysing the memory utilisation of the protocol. The number of states required for leader election and buffering is $O(\log n)$, as shown in Lemma 10 and Lemma 21 of [25], respectively. However, agents occupying states used for leader election and buffering do not participate in the actual localisation process. As a result, the overall state space of the self-stabilising protocol is dominated by that of Algorithm 1, extended by a deadline counter of size $O\left(n(\log n/n)^{1/(k+1)}\right)$ used by the `blue` agents.

3.2 Leader election

We begin by presenting a suitable leader election protocol that operates in $O(\log n)$ time and uses $O(\log n)$ space. This protocol is executed at the beginning of the first stage of each round and is followed by the main localisation protocol, Algorithm 1, augmented with anomaly detection and deadline counters.

Leader election is done by independently assigning a leader role to each agent with probability $p = \Theta(1/n)$. In this way, the probability of electing exactly one leader is constant and is maximised when $p = 1/n$ and is then approximately $1/e$. In our protocol, each agent independently tosses a symmetric coin $\log n$ times and becomes the leader if it gets heads on all tosses. We are left to describe the protocol implementing this process.

States in this protocol are denoted as pairs (A, i), in which $A \in \{N, H, T\}$ and $i \in \{1, \ldots, \log n, L, F\}$. The first component, A, represents the type of coin toss state: *Neutral* (N), *Head* (H), or *Tails* (T). The second component, i, either serves as a counter indicating progress toward successful leader election, or denotes a status: L for leader and F for follower. On leaving the buffering mechanism (the second stage of the previous round) all agents receive state $(N, 1)$, and the only meaningful interactions in the leader election protocol are:

- The creation of $H : T$ state pairs ensuring that there are consistently the same number of agents with H and T states in the population: $(N, *) + (N, *) \to (H, *) + (T, *)$.
- A coin toss in which the initiator gets heads: $(*, i) + (H, *) \to (*, i + 1) + (H, *)$, if $i < \log n$, and $(*, \log n) + (H, *) \to (*, L) + (H, *)$, otherwise.
- A coin toss in which the initiator gets tails: $(*, i) + (T, *) \to (*, F) + (T, *)$, if $i \in \{1, \ldots, \log n\}$.

▶ **Lemma 10.** *A unique leader is elected in $O(\log n)$ time with a constant probability.*

Proof. Let us assume that at time 0 all agents have already left the buffer and joined the leader election protocol. For as long as the number of agents in $(N, *)$ states exceeds $n/2$ the probability of forming a $H : T$ pair in a given interaction exceeds $1/4$. Thus, the expected time in which the number of agents in $(N, *)$ states falls below $n/2$ is less than 2, coinciding with $2n$ interactions. And whp this time is less than 3. After this time, the probability that an agent with a counter in an interaction makes a coin flip is $1/2n$. Thus, in a time period of $2c \cdot \log n$ it performs at least $c \cdot \log n$ coin tosses on average. From Chernoff's inequality for a sufficiently large constant c it makes at least $c \cdot \log n$ coin tosses whp. From the union bound in total time $O(\log n)$ all agents will perform $\log n$ coin tosses each, which will establish their leader (L) or follower (F) status. ◀

We still need to explain how to adopt this leader election protocol in the self-stabilising leader-based localisation protocol. Later in Section 3.4, we show that at some point all agents are located in the buffer, and ready to proceed to the next round. When the first agent leaves the buffer, it assumes the state $(N, 1)$, which triggers an epidemic process that informs all other agents in the buffer to adopt the same state $(N, 1)$. Note that all agents participating in the leader election protocol contribute to this epidemic, ensuring that the entire population engages in leader election within $O(\log n)$ time. Upon completing the leader election, agents with confirmed leader or follower status start leader-based localisation protocol adopting `green` and `blue` states, respectively. In addition, they never give up their coin ($H|T|N$) attributes to guarantee fairness of the remaining coin tosses for agents still executing leader election protocol. Thus, after integration, the overall running time of the leader election protocol remains $O(\log n)$.

3.3 Augmented leader-based localisation

On the conclusion of the leader election process leader-based localisation (Algorithm 1) is executed, however, with a few modifications. Namely, the interaction counter with the deadline $O\left(n(\log n/n)^{1/(k+1)}\right)$ is added to every new `blue` agent. This is to ensure that the first stage does not exceed the expected stabilisation time of Algorithm 1. One also needs to

manage anomaly detection. Recall, that two types of anomalies may arise. The first, *label inconsistency*, occurs only between two `green` agents. In particular, when all agents become `green`, this type of anomaly is detected in time $O(k \log n)$, see the lemma below.

▶ **Lemma 11.** *When all agents are `green`, the anomaly based on labels inconsistency is detected in time $O(k \log n)$ whp.*

Proof. It is known that a random graph with nodes embedded in k-dimensional space and $O(k \log n)$ edges chosen uniformly at random is globally rigid whp [36]. This implies that there exists a unique embedding of the graph resulting from the interactions, up to isometries such as rotations, translations, and reflections. Consequently, if the labels of any `green` agents are inconsistent, such inconsistencies will be detected after $O(k \log n)$ interactions. ◀

The second anomaly occurs when any `blue` agent's counter reaches the deadline. Upon detecting either of the anomalies, a reset signal is triggered initiating the buffering mechanism discussed in the next section.

3.4 Buffering mechanism

The buffering for reset of the self-stabilising localisation protocol is solved in the same way as the buffering in the ranking protocol with $O(\log n)$ additional states from [25]. In this work, the signal for reset is the detection of an anomaly. It causes the state of the agents that performed the anomaly detection to be set to the first state of the buffer: X_1. The buffer is a line consisting of $2d \log n$ states: $X_1, X_2, \ldots, X_{2d \log n}$ for d large enough. We assign `red` colour to the states $X_1, \ldots, X_{d \log n}$ and `white` (`green` in [25]) to the remaining buffer states.

We define the following transitions for these states:

- *Progress* on the line: $X_i + X_j \to X_{i+1} + X_{i+1}$ for $i \leq j$.
- *Reset propagation* by `red` agents: $X_i + A \to X_1 + X_1$, if $i \leq d \log n$ and A not on the line.
- *Buffer departure* by `white` agents: $X_{2d \log n} + X_{2d \log n} \to (N, 1) + (N, 1)$, where $(N, 1)$ is the initial state of the leader election phase. Also $X_i + A \to (N, 1) + A$, where $i > d \log n$ and A is not on the line.

The analysis of the buffering process is summed up in the paper [25] by the following Lemma, proved there

▶ **Lemma 12** (Lemma 21 in paper [25])**.** *There exists $d' > 0$, s.t., for any $d \geq 0$ there exists $c > 0$, for which after at most $c \log n$ time since state X_1 (reset signal) arrival, all agents are in line states X_i with indices $d \log n < i \leq (d + d') \log n$ whp.*

If we substitute our d for d and d' in this lemma we obtain that at time $O(\log n)$ after the first agents appear in state X_1, all agents are `white`. On the other hand, when we substitute $2d$ for d in the lemma and d for d', we obtain that at time $O(\log n)$ the agents start to leave the line of the buffer.

We conclude Section 3 with the following theorem.

▶ **Theorem 13.** *The localisation protocol presented in this section self-stabilises silently in time $O(n(\log n/n)^{1/(k+1)} \log n)$ whp.*

4 Self-stabilising localisation in vector query model

In this section, we adopt the *vector query model*, assuming first that S is a linear space where agents $A_0, \ldots, A_{n-1}$ are arbitrarily positioned at (unknown to the agents) points $p_0, \ldots, p_{n-1}$, respectively. We use notation $\overrightarrow{v_{ij}}$ to denote the vector connecting p_i and p_j, in

this case $\overrightarrow{v_{ij}} = p_j - p_i$. Additionally, each agent A_i possesses a *hypothetical coordinate* referred to as its *label* x_i. In this variant of population protocols, during an interaction of A_i with A_r, the initiator A_i learns about vector $\overrightarrow{v_{ir}}$ and label x_r. We show that this model is very powerful as it allows design of an optimal $O(\log n)$ time self-stabilising labelling protocol.

Algorithm 2 Fast positioning in one dimension.

```
/* Code executed by initiator A_i interacting with A_r                    */
local state     : x_i
received data : x_r, v_ir
1 begin
2 |  x_i ← max{x_i, x_r − v_ir}
3 end
```

We start with a trivial fact concerning label updates.

▶ **Fact 1.** *For each agent A_i its label x_i never decreases during the execution of Algorithm 2.*

Let $M = \max_{0 \le i \le n-1} (x_i - p_i)$.

▶ **Fact 2.** *The value of M does not change during the execution of Algorithm 2.*

Proof. Consider an effective update of label x_i, where x_i' is the updated value of this label. $x_i' = x_r - \overrightarrow{v_{ir}} = x_r - (p_r - p_i)$. Thus, $x_i' - p_i = x_r - p_r \le M$. ◀

After this update agent A_i adopts new label $x_i = x_r - p_r$. Let $S_M = \{A_i : x_i - p_i = M\}$.

▶ **Fact 3.** *For any two agents $A_i, A_j \in S_M$ we have $x_j - x_i = p_j - p_i$.*

Proof. Note that $\overrightarrow{v_{ij}} = p_j - p_i = x_j - M - (x_i - M) = x_j - x_i$. ◀

Thus, the labels of agents in S_M are stable, meaning they align with the coordinate system that will ultimately be agreed upon by all agents.

▶ **Fact 4.** *After $A_i \notin S_M$ interaction with $A_r \in S_M$, A_i becomes an element of S_M.*

Proof. As $A_i \notin S_M$, we get $x_i - p_i < M$. However, as $r \in S_M$ we also get $x_i - p_i < x_r - p_r$, and in turn $x_i < x_r - p_r + p_i < x_r - \overrightarrow{v_{ir}}$. Thus, x_i becomes $x_r - \overrightarrow{v_{ir}}$, which is equal to M as $r \in S_M$. ◀

▶ **Theorem 14.** *The localisation Algorithm 2 self-stabilises silently in the optimal time $O(\log n)$ whp.*

Proof. The membership of agents in S_M is spread via one-way epidemic in time $O(\log n)$ whp. This protocol is silent as after all agents are included in S_M, further label updates are not possible. Ultimately, this protocol is self-stabilising, requiring no initial assumptions about agents' labels, and achieves time optimality of $O(\log n)$, corresponding to the communication bound in population protocols. ◀

Algorithm 2 can be extended to higher fixed dimensions by applying multiple instances of the one-dimensional protocol to each dimension's coordinates, as shown in Algorithm 3.

▶ **Theorem 15.** *The localisation Algorithm 3 self-stabilises silently in a fixed k dimensional space in the optimal time $O(\log n)$ whp.*

Proof. By directly applying the union bound. ◀

Algorithm 3 Fast positioning in fixed k dimensions.

```
/* Code executed by initiator A_i interacting with A_r                    */
local state     : x_i = [x_i[0], ..., x_i[k-1]]
received data : x_r, v_ir
1 begin
2 |  foreach j ∈ [0, k-1] do
3 |  |  x_i[j] ← max{x_i[j], x_r[j] − v_ir[j]}
4 |  end
5 end
```

5 Concluding remarks

In this paper, we introduce a novel variant of spatial population protocols and explore its applicability to the distributed localisation problem. Any meaningful advances in this problem could pave the way for developing faster and more robust lightweight communication protocols suitable for real-world applications. It could also provide insights into the limitations of what can be achieved in such systems.

Several challenges remain unresolved in this work. Firstly, we do not account for inaccuracies in distance measurements. As no measuring device is perfect, future studies should model measurement errors, which can significantly affect system stability and performance. Our preliminary studies on the fast positioning protocol from Section 4 reveal a phenomenon called label *drifting* in the presence of errors, resembling phase clock behaviour. This drifting can be controlled to achieve near-complete label stabilisation and may support mobility coordination in large robotic agent populations.

Second, we leave the issue of limited communication range unaddressed. In real-world scenarios, agents often cannot communicate with all other agents in the system, which adds further complexity. As mentioned in the introduction, it is well known that limited communication range and arbitrary network topologies can lead to intractable localisation problems in the worst case. Therefore, a promising direction for future research would be to focus on specific classes of network topologies for which lightweight localisation protocols are more likely to be effective.

A third challenge is the development of efficient localisation algorithms for mobile agents. In this case, assumptions about the relative speeds of communication and movement would likely be necessary to ensure that data from previous positions can still be utilised effectively.

Finally, of independent interest are further studies on the computational power of spatial population protocols, both in comparison to existing variants of population protocols and in relation to various geometric problems, types of queries, distance-based biased communication, and other related topics.

References

1 D. Alistarh, J. Aspnes, D. Eisenstat, R. Gelashvili, and R.L. Rivest. Time-space trade-offs in population protocols. In *SODA 2017*, pages 2560–2579, 2017.

2 D. Alistarh, J. Aspnes, and R. Gelashvili. Space-optimal majority in population protocols. 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 2221–2239. SIAM, 2018. doi:10.1137/1.9781611975031.144.

3 D. Alistarh and R. Gelashvili. Polylogarithmic-time leader election in population protocols. In M.M. Halldórsson, K. Iwama, N. Kobayashi, and B. Speckmann, editors, *ICALP 2015, Part II*, pages 479–491, 2015.

4 D. Alistarh, R. Gelashvili, and J. Rybicki. Fast Graphical Population Protocols. In *OPODIS 2021*, pages 14:1–14:18, 2022. doi:10.4230/LIPIcs.OPODIS.2021.14.

5 D. Alistarh, J. Rybicki, and S. Voitovych. Near-optimal leader election in population protocols on graphs. In *PODC'22*, pages 246–256, 2022.

6 D. Angluin, J. Aspnes, Z. Diamadi, M.J. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. In *PODC 2004*, pages 290–299, 2004.

7 D. Angluin, J. Aspnes, and D. Eisenstat. Fast computation by population protocols with a leader. *Distributed Comput.*, 21(3):183–199, 2008. doi:10.1007/S00446-008-0067-Z.

8 J. Aspnes, D.K. Goldenberg, and Y.R. Yang. On the computational complexity of sensor network localization. In *ALGOSENSORS 2004*, pages 32–44, 2004.

9 J. Beauquier, P. Blanchard, and J. Burman. Self-stabilizing leader election in population protocols over arbitrary communication graphs. In *OPODIS 2013*, pages 38–52, 2013.

10 P. Berenbrink, R. Elsässer, T. Götte, L. Hintze, and D. Kaaser. Silent self-stabilizing ranking: Time optimal and space efficient. *CoRR*, abs/2504.10417, 2025. To appear at ICDCS 2025. doi:10.48550/arXiv.2504.10417.

11 P. Berenbrink, G. Giakkoupis, and P. Kling. Optimal time and space leader election in population protocols. In *STOC 2020*, pages 119–129, 2020.

12 P. Berenbrink, D. Kaaser, P. Kling, and L. Otterbach. Simple and efficient leader election. In *SOSA 2018*, volume 61 of *OASIcs*, pages 9:1–9:11, 2018. doi:10.4230/OASICS.SOSA.2018.9.

13 A. Bilke, C. Cooper, R. Elsässer, and T. Radzik. Brief announcement: Population protocols for leader election and exact majority with $O(\log^2 n)$ states and $O(\log^2 n)$ convergence time. In *PODC 2017*, pages 451–453, 2017.

14 J. Burman, H.-L. Chen, H.-P. Chen, D. Doty, T. Nowak, E.E. Severson, and C. Xu. Time-optimal self-stabilizing leader election in population protocols. In *PODC 2021*, pages 33–44, 2021.

15 S. Cai, T. Izumi, and K. Wada. How to prove impossibility under global fairness: On space complexity of self-stabilizing leader election on a population protocol model. *Theory of Computer Systems*, 50(3):433–445, 2012. doi:10.1007/S00224-011-9313-Z.

16 H.-L. Chen, R. Cummings, D. Doty, and D. Soloveichik. Speed faults in computation by chemical reaction networks. In Fabian Kuhn, editor, *DISC 2014*, pages 16–30, 2014.

17 P. Damaschke. Point placement on the line by distance data. *Discret. Appl. Math.*, 127(1):53–62, 2003. doi:10.1016/S0166-218X(02)00284-6.

18 P. Damaschke. Randomized vs. deterministic distance query strategies for point location on the line. *Discret. Appl. Math.*, 154(3):478–484, 2006. doi:10.1016/J.DAM.2005.07.014.

19 D. Doty. Timing in chemical reaction networks. In *SODA 2014*, pages 772–784, 2014.

20 D. Doty and D. Soloveichik. Stable leader election in population protocols requires linear time. *Distributed Comput.*, 31(4):257–271, 2018. doi:10.1007/S00446-016-0281-Z.

21 D. Doty and D. Soloveichik. Stable leader election in population protocols requires linear time. *Distributed Computing*, 31(4):257–271, 2018. doi:10.1007/S00446-016-0281-Z.

22 T. Eren, D.K. Goldenberg, W. Whiteley, Y.R. Yang, A.S. Morse, B.D.O. Anderson, and P.N. Belhumeur. Rigidity, computation, and randomization in network localization. In *INFOCOM 2004*, pages 2673–2684, 2004.

23 R. Fareh, M. Baziyad, S. Khadraoui, B. Brahmi, and M. Bettayeb. Logarithmic potential field: A new leader-follower robotic control mechanism to enhance the execution speed and safety attributes. *IEEE Access*, 2023.

24 P. Flocchini, G. Prencipe, and N. Santoro, editors. *Distributed Computing by Mobile Entities, Current Research in Moving and Computing*, volume 11340 of *LNCS*. Springer, 2019. doi:10.1007/978-3-030-11072-7.

25 L. Gąsieniec, T. Grodzicki, and G. Stachowiak. Near-state and state-optimal self-stabilising leader election population protocols. *CoRR*, abs/2502.01227, 2025. To appear at PODC 2025. doi:10.48550/arXiv.2502.01227.

26 L. Gąsieniec and G. Stachowiak. Enhanced phase clocks, population protocols, and fast space optimal leader election. *J. ACM*, 68(1):2:1–2:21, 2021. doi:10.1145/3424659.

27 L. Gąsieniec, G. Stachowiak, and P. Uznański. Almost logarithmic-time space optimal leader election in population protocols. In *SPAA 2019*, pages 93–102, 2019.

28 B. Hendrickson. Conditions for unique graph realizations. *SIAM Journal on Computing*, 21(1):65–84, 1992. doi:10.1137/0221008.

29 J. Lagergren J. Håstad, L. Ivansson. Fitting points on the real line and its application to rh mapping. *J. Algorithms*, 49:42–62, 2003. doi:10.1016/S0196-6774(03)00083-X.

30 W.L. Ruzzo J. Redstone. Algorithms for a simple point placement problem. In *CIAC 2000*, volume 1767 of *Lecture Notes in Computer Science*, pages 32–43, 2000. doi:10.1007/3-540-46521-9_3.

31 B. Jackson and T. Jordán. Connected rigidity matroids and unique realizations of graphs. *J. Comb. Theory B*, 94(1):1–29, 2005. doi:10.1016/J.JCTB.2004.11.002.

32 E. Latif and R. Parasuraman. Dgorl: Distributed graph optimization based relative localization of multi-robot systems. In *International Symposium on Distributed Autonomous Robotic Systems*, pages 243–256. Springer, 2022.

33 E. Latif and R. Parasuraman. Multi-robot synergistic localization in dynamic environments. In *ISR Europe 2022; 54th International Symposium on Robotics*, pages 1–8. VDE, 2022.

34 E. Latif and R. Parasuraman. Instantaneous wireless robotic node localization using collaborative direction of arrival. *IEEE Internet of Things Journal*, 2023.

35 E. Latif and R. Parasuraman. GPRL: Gaussian processes-based relative localization for multi-robot systems. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2024.

36 A. Lew, E. Nevo, Y. Peled, and O.E. Raz. Sharp threshold for rigidity of random graphs. *Bulletin of the London Mathematical Society*, 55(1):490–501, 2023.

37 A.P. Moreira, P. Costa, and J. Lima. New approach for beacons based mobile robot localization using Kalman filters. *Procedia Manufacturing*, 51:512–519, 2020.

38 B. Mumey. Probe location in the presence of errors: a problem from dna mapping. *Discrete Appl. Math.*, 104:187–201, 2000. doi:10.1016/S0166-218X(00)00189-X.

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

40 Y. Sudo, F. Ooshita, H. Kakugawa, T. Masuzawa, A.K. Datta, and L.L. Larmore. Loosely-stabilizing leader election with polylogarithmic convergence time. *Theoretical Computer Science*, 806:617–631, 2020. doi:10.1016/J.TCS.2019.09.034.

41 S. Wang, Y. Wang, D. Li, and Q. Zhao. Distributed relative localization algorithms for multi-robot networks: A survey. *Sensors*, 23(5):2399, 2023. doi:10.3390/S23052399.

42 J. X. Guo, Hu, J. Chen, F. Deng, and T.L. Lam. Semantic histogram based graph matching for real-time multi-robot global localization in large scale environment. *IEEE Robotics and Automation Letters*, 6(4):8349–8356, 2021. doi:10.1109/LRA.2021.3058935.

43 M. Xu, N. Snderhauf, and M. Milford. Probabilistic visual place recognition for hierarchical localization. *IEEE Robotics and Automation Letters*, 6(2):311–318, 2020.

44 D. Yokota, Y. Sudo, and T. Masuzawa. Time-optimal self-stabilizing leader election on rings in population protocols. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, 104-A(12):1675–1684, 2021. doi:10.1587/TRANSFUN.2020EAP1125.

45 D. Yokota, Y. Sudo, F. Ooshita, and T. Masuzawa. A near time-optimal population protocol for self-stabilizing leader election on rings with a poly-logarithmic number of states. In *PODC 2023*, pages 2–12, 2023.

46 Y. Yue and D. Wang. *Collaborative Perception, Localization and Mapping for Autonomous Systems*, volume 2. Springer Nature, 2020.

Streaming Periodicity with Mismatches, Wildcards, and Edits

Taha El Ghazi
DIENS, École normale supérieure de Paris, PSL Research University, France

Tatiana Starikovskaya
DIENS, École normale supérieure de Paris, PSL Research University, France

Abstract

In this work, we study the problem of detecting periodic trends in strings. While detecting exact periodicity has been studied extensively, real-world data is often noisy, where small deviations or mismatches occur between repetitions. This work focuses on a generalized approach to period detection that efficiently handles noise. Given a string S of length n, the task is to identify integers p such that the prefix and the suffix of S, each of length $n - p + 1$, are similar under a given distance measure. Ergün et al. [APPROX-RANDOM 2017] were the first to study this problem in the streaming model under the Hamming distance. In this work, we combine, in a non-trivial way, the Hamming distance sketch of Clifford et al. [SODA 2019] and the structural description of the k-mismatch occurrences of a pattern in a text by Charalampopoulos et al. [FOCS 2020] to present a more efficient streaming algorithm for period detection under the Hamming distance. As a corollary, we derive a streaming algorithm for detecting periods of strings which may contain wildcards, a special symbol that match any character of the alphabet. Our algorithm is not only more efficient than that of Ergün et al. [TCS 2020], but it also operates without their assumption that the string must be free of wildcards in its final characters. Additionally, we introduce the first two-pass streaming algorithm for computing periods under the edit distance by leveraging and extending the Bhattacharya–Koucký's grammar decomposition technique [STOC 2023].

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases approximate periods, pattern matching, streaming algorithms

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.36

Related Version *Full Version*: https://arxiv.org/abs/2509.14898

Funding Partially supported by the grant ANR-20-CE48-0001 from the French National Research Agency (ANR) and a Royal Society International Exchanges Award.

1 Introduction

In this work, we consider the problem of computing periodic trends in strings. Informally, a string has period p if its prefix of length $n - p + 1$ equals its suffix of length $n - p + 1$. Alternatively, a string has period p if it equals its prefix of length p repeated a (possibly, fractional) number of times. The problem of detecting periodic trends in strings has numerous practical applications, including fields like astronomy, financial analytics, and meteorology (see e.g. [11]). The nature and the volume of the data in the applications ask for algorithms able to process the input in very little space and in (almost) real time, such as streaming algorithms. In the streaming setting, we assume that the input string arrives one character at a time and account for all space used, even for the space used to store information about the input data, which results in ultra-efficient algorithms.

The first streaming algorithm for detecting exact periods of a stream of length n was given by Ergün, Jowhari, and Sağlam [14], who presented an algorithm with $O(\log^2 n)$ space.[1]

[1] Hereafter, the space is measured in bits.

© Taha El Ghazi and Tatiana Starikovskaya;
licensed under Creative Commons License CC-BY 4.0

While the problem of detecting exact periodicity is fundamental to string processing, in practice data is rarely exactly periodic. For example, a string *abcabcabaabaaba* is clearly repetitive, but does not have a small exact period in the sense above. To account for these variations in natural data, Ergün, Jowhari, and Sağlam [14] proposed another streaming algorithm, which allows for approximately computing the Hamming distance (the number of mismatches) between a stream and a string with period p, for a given integer p. Their algorithm computes the distance with an approximation factor $2 + \varepsilon$ in space $\tilde{O}(1/\varepsilon^2)$. A disadvantage of this algorithm is that p must be known in advance, and knowing p approximately will not suffice.

Later, Ergün, Grigorescu, Azer, and Zhou [12] suggested a different version of this problem: given a string of length n and an integer k, they asked for all integer p such that the Hamming distance between the string and its copy shifted by p is at most k (see Section 2 for a definition). They called such p *k-mismatch periods*. They proved that for computing all k-mismatch periods of a streaming string of length n one needs $\Omega(n)$ space and that for computing all k-mismatch periods in $[1..n/2]$, one needs $\Omega(k \log n)$ space. On the other hand, prior work [12, 15] claimed a streaming algorithm that, given a string of length n, computes all its k-mismatch periods in $[1..n/2]$ using only $O(k^4 \log^9 n)$ space. However, we found significant gaps in the proof of correctness. Many key arguments are only briefly outlined in both the conference version [12] and the longer arXiv manuscript [15], and as confirmed in private communication with the authors, a full journal version of the paper has not been published. In particular, [15, Theorem 28] is proved only for the case when the string has two distinct k-mismatch periods, and is stated to generalize easily to the case when the string has more than two distinct periods. We believe this generalization does not hold; see Appendix A for a detailed explanation.

Finally, in a different work [13] Ergün, Grigorescu, Azer, and Zhou considered the problem of computing periods of incomplete streaming data, where some characters are replaced with wildcards, a special symbol that matches each character of the alphabet. We say that a wildcard-containing string T has an integer period p if T shifted by p positions matches itself. They showed that a streaming algorithm which computes all periods of an n-length string with wildcards requires $\Omega(n)$ space. Conversely, they demonstrated a streaming algorithm which given a string T with k wildcards computes all its periods $p \leq n/2$ under condition that there are no wildcards in the last p characters of T. The algorithm uses $O(k^3 \operatorname{polylog} n)$ space. They also showed that for $k = o(\sqrt{n})$, such an algorithm must use $\Omega(k \log n)$ space.

Our results

In this work, we continue the study initiated by Ergün, Grigorescu, Azer, and Zhou [12]. We first give a streaming algorithm for computing the k-mismatch periods of a string (Theorem 2.10), providing full details and improving over the claimed space bound of Ergün, Grigorescu, Azer, and Zhou [12] by a factor of $k^2 \log^5 n$.

As an almost immediate corollary, and our second contribution, we obtain a similar result for computing periods of a string with few wildcards (Theorem 2.2). Specifically, we both improve the complexity of the algorithm by Ergün, Grigorescu, Azer, and Zhou [13] by a factor of $k \operatorname{polylog} n$, narrowing the gap between the upper and lower bounds, and remove the assumption that there are no wildcards in the last characters of the input string.

As our third and final contribution, we extend our results to the edit distance. Informally, we say that a string has a *k-edit period* $p \leq n/2$ if the edit distance (the number of deletions, insertions, and substitutions required to transform one string into another) between the input string and its copy shifted by p characters is at most k (see Section 2 for a formal

definition). We apply the Bhattacharya–Koucký's grammar decomposition [4] and then show that we can compute the k-edit periods of the input by computing the k-mismatch periods of the stream of suitably encoded grammars. As our algorithm for k-mismatch periods needs to know the length of the stream (assumption already present in [12]), we make two passes: first, we compute the decomposition and the length of the stream, and in the second pass we compute the periods. As a result, we develop the first two-pass streaming algorithm for computing k-edit periods of a string in $\tilde{O}(k^2 n)$ time and $\tilde{O}(k^4)$ space (Theorem 2.3).[2] We emphasize that in this work, we deliberately chose to treat our algorithm for k-mismatch periods as a black box, using it as a direct application in the context of k-edit periods. While it is conceivable that opening this black box and combining its internal components with the Bhattacharya–Koucký technique could lead to a one-pass streaming algorithm for k-edit periods, pursuing this direction would require significant new insights. We leave this as a promising open question for future research.

Related work

Other repetitive string structures that have been considered in the literature include palindromes (strings that read the same forward and backwards) and squares (strings equal to two copies of a string concatenated together). Berebrink et al. [3] followed by Gawrychowski et al. [16] studied the question of computing the length of a maximal palindromic substring of a stream that is a palindrome. Merkurev and Shur [21] considered a similar question for squares. Bathie et al. [1] considered the problem of recognising prefixes of a streaming string that are at Hamming (edit) distance at most k from palindromes and squares.

2 Results and technical overview

In this work, we assume the word-RAM model of computation and work in a particularly restrictive streaming setting. In this setting, we assume that the input string arrives as a stream, one character at a time. We must account for all the space used, including the space used to store information about the input. Throughout, the space is measured in bits.

Notations and terminology

We assume to be given an alphabet Σ, the elements of which, called *characters*, can be stored in a single machine word of the word-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 $S^0 = \varepsilon$. Furthermore, S^∞ denotes an infinite string obtained by concatenating infinitely many copies of S. For a string $T \in \Sigma^n$ and $i \in [1..n]$,[3] the ith character of T is denoted by $T[i]$. We use $|T| = n$ to denote the length of T. For $1 \leq i, j \leq n$, $T[i..j]$ 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 them, i.e., we write $T[..j] = T[1..j]$ and $T[i..] = T[i..n]$. 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 $P = T[i..]$. A non-empty string $T \in \Sigma^n$ is *primitive* if $T^2[i..j] = T$ implies $i = 1$ or $i = |T| + 1$.

[2] Hereafter, $\tilde{O}$ means that we hide factors polylogarithmic in n. "Two-pass" means that the algorithm reads T as a stream twice.

[3] 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 $(i..j] = \{k \in \mathbb{Z} : i < k \leq j\}$.

2.1 k-mismatch periods

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 $\{(i, S[i], T[i]) : i \in [1..n] \text{ and } S[i] \neq T[i]\}$. An integer p is called a *k-mismatch period* of a string T of length n if $1 \leq p \leq n$ and $\mathsf{hd}(T[1..n-p+1], T[p..n]) \leq k$.

▶ **Theorem 2.1** (Informal statement of Theorem 2.10)**.** *Given a string T of length n and an integer k, there is a streaming algorithm that computes all k-mismatch periods of T in $[1..n/2]$. Furthermore, for each detected k-mismatch period p, it also returns $\mathsf{MI}(T[p..], T[..n-p+1])$. The algorithm uses $O(n \cdot k \log^5 n)$ time and $O(k^2 \log^3 n)$ space, and is correct w.h.p.*[4]

To develop our result, we start with a simple idea already present in [12]: If p is a k-mismatch period of T, then for all $1 \leq \ell \leq n-p$, the position p is a starting position of a k-mismatch occurrence of $T[1..\ell+1]$. This already allows to filter out candidate periods. To test whether an integer p is a k-mismatch period, we check if the Hamming distance between $T[1..n-p+1]$ and $T[p..n]$ is at most k. For this, we aim to use the Hamming distance sketches introduced by Clifford, Kociumaka, and Porat [9] for strings $T[1..n-p+1]$ and $T[p..n]$. There are two main challenges: First, we might discover a candidate k-mismatch period p after passing position $n-p+1$, which would make it impossible to compute the Hamming distance sketch of $T[1..n-p+1]$ due to streaming access limitations. To address this, as in [12], we divide the interval $[1..n/2]$ into a logarithmic number of subintervals, filtering positions in each subinterval using progressively shorter prefixes. The second challenge is that the number of candidate periods can be large, and we cannot store all the sketches. Here, we diverge significantly from [12]: To store the sketches in small space, we leverage the concatenability of the Hamming distance sketches of [9] and the structural regularity of k-mismatch occurrences as shown by Charalampopoulos, Kociumaka, and Wellnitz [8]. Combining the two ideas together is non-trivial and this is where the main novelty of our result for the Hamming distance is. The proof of Theorem 2.10 is given in Section 3.

Assume to be given a string containing wildcards. By replacing wildcards in a string with a new character, we immediately derive the following:

▶ **Theorem 2.2.** *Given a string T of length n containing at most k wildcards. There is a streaming algorithm that computes the set of periods of T in $[1..n/2]$ in $O(n \cdot k \log^5 n)$ time and $O(k^2 \log^3 n)$ space. The algorithm is correct w.h.p.*

Proof. If we replace the wildcards in T with a new character $\# \notin \Sigma$, then a period p of T is a k-mismatch period of the resulting string in the alphabet $\Sigma \cup \{\#\}$, and we can find it via Theorem 2.1. To check that the positions returned by the algorithm are periods of the original string, for each detected k-mismatch period, we retrieve the relevant mismatch information and verify that for all mismatching pairs of characters, at least one of those is the special character $\#$. ◀

2.2 k-edit periods

The edit distance between two strings S, T (denoted by $\mathsf{ed}(S,T)$) is the minimum number of character insertions, deletions, and substitutions required to transform S into T. We say that an integer p is a *k-edit period* of a string T of length n if $1 \leq p \leq |T|$ and for some $1 \leq i \leq n$, $\mathsf{ed}(T[1..i], T[p..n]) \leq k$.

[4] Hereafter, w.h.p. stands for probability at least $1 - 1/n^c$, for a constant $c > 1$.

▶ **Theorem 2.3.** *Given a string T of length n and an integer k, there is a two-pass streaming algorithm that computes all k-edit periods of T in $\tilde{O}(k^2 n)$ time and $\tilde{O}(k^4)$ space. The algorithm is correct w.h.p.*

2.2.1 Preliminaries: Grammar decomposition

We assume familiarity with the notion of straight-line programs (SLP) [18], which represent a subclass of context-free grammars. The size of an SLP G is the number of non-terminals and is denoted by $|G|$. Furthermore, G represents a unique string, called the *expansion* of G, $\mathsf{exp}(G)$. The length of $|\mathsf{exp}(G)|$ can be computed in $O(|G|)$ time and space [20].

▶ **Fact 2.4.** *Let G_X and G_Y be SLPs representing strings X and Y respectively, and $m = |G_X| + |G_Y|$. Both of the following hold:*

- *We can compute $d := \mathsf{ed}(X, Y)$ in $O((m + d^2)\log|XY|)$ time and $O(m \log|XY|)$ space (see [5, Proposition 2.1]).*
- *Given an integer k, we can find all $1 \le i \le |Y|$ such that $\mathsf{ed}(X, Y[i..]) \le k$ and the corresponding edit distances in $O((m + k^2)\log|XY|)$ time and $O(m \log|XY|)$ space.*

Proof. By [19] (see also the remark at the end of [10, Section 5]), all such positions i can be found in $(k+1)^2$ *longest common extension* (LCE) queries on a string Z equal to the reverse of XY. Given two positions $1 \le i, j \le |Z|$, an LCE query asks for the maximal ℓ such that $Z[i..i+\ell-1] = Z[j..j+\ell-1]$. The string Z can be represented as the expansion of an SLP of size $O(|G_X| + |G_Y|)$. I [17] showed that after $O(m \log|Z|)$-time and -space preprocessing of the SLP[5], LCE queries on Z can be answered in $O(\log|Z|)$ time. The claim follows. ◀

One of the central tools of our solution is Bhattacharya–Koucký's grammar decomposition (BK-decomposition) [4]. It is a randomised decomposition that uses as source of randomness two families of hash functions $C_1, \ldots, C_L$ and $H_0, \ldots, H_L$, where $L = O(\log n)$ is a suitably chosen parameter. The decomposition of a string X is a sequence $\mathcal{G}(X)$ of SLPs.[6] For a sequence of SLPs $\mathcal{G} = G_1 \cdots G_m$, define $\mathsf{exp}(\mathcal{G}) = \mathsf{exp}(G_1) \cdots \mathsf{exp}(G_m)$.

▶ **Fact 2.5** ([4, Theorem 3.1])**.** *Let $X \in \Sigma^{\le n}$, $k \in \mathbb{N}$ be the input parameter of the grammar decomposition algorithm, and let $\mathcal{G}(X) = G_1^X \cdots G_s^X$. For all n large enough, $X = \mathsf{exp}(\mathcal{G}(X))$ and $|G_i^X| = \tilde{O}(k)$ for $i \in \{1, \ldots, s\}$ with probability $\ge 1 - 2/n$.*

The BK-decomposition can be maintained in a streaming fashion:

▶ **Corollary 2.6** ([6, Lemma 4.2,Theorem 5.1])**.** *Given a streaming string $X \in \Sigma^*$, there is an algorithm that outputs a stream of SLPs $\mathcal{G}_{\text{def}} = G_1 \cdots G_s$ (referred to as* definite*) and maintains a sequence $\mathcal{G}_{\text{active}} = G'_1 \cdots G'_t$ of $\tilde{O}(1)$ SLPs (referred to as* active*) such that after having read $X[1..i]$, the concatenation of $\mathcal{G}_{\text{def}}$ and $\mathcal{G}_{\text{active}}$ equals $\mathcal{G}(X[1..i])$. The algorithm uses $\tilde{O}(k)$ time per character and $\tilde{O}(k)$ space (here, we account for all the space used except for the space required to store $\mathcal{G}_{\text{def}}$). The only operation the algorithm is allowed to perform on $\mathcal{G}_{\text{def}}$ is appending a grammar from the right.*

We further make use of the following claim:

[5] The result of I [17] gives tighter bounds, but this is sufficient for our purposes

[6] Strictly speaking, the decomposition [4] outputs run-length SLPs. However, one can transform those grammars into SLPs with a size blow-up polylogarithmic in n [20].

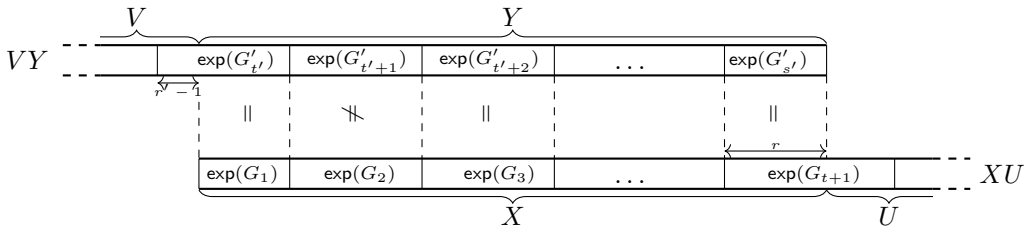


Figure 1 Grammar decompositions for VY and XU for the case $\mathsf{ed}(X,Y) \leq k$.

▶ **Corollary 2.7.** *Let $k \leq n$ be integers. Assume $X,Y,U,V \in \Sigma^*$, $|XU|,|VY| \leq n$, and $\mathsf{ed}(X,Y) \leq k$. Let $\mathcal{G}(XU) = G_1 \cdots G_s$ and $\mathcal{G}(VY) = G'_1 \cdots G'_{s'}$. With probability at least $1-1/5$, there exist integers r, r', t, t' such that each of the following is satisfied (see Figure 1):*

1. $t+1 = s'-t'+1$.
2. $X = \mathsf{exp}(G_1 \cdots G_t) \cdot \mathsf{exp}(G_{t+1})[..r]$ *and* $Y = \mathsf{exp}(G'_{t'})[r'..] \cdot \mathsf{exp}(G'_{t'+1} \cdots G'_{s'})$.
3. $G_i = G'_{t'+i-1}$ *except for at most $k+2$ indices $1 \leq i \leq t+1$.*
4. $\mathsf{ed}(X,Y)$ *equals the sum of* $\mathsf{ed}(\mathsf{exp}(G_1), \mathsf{exp}(G'_{t'})[r'..])$, $\sum_{2 \leq i \leq t} \mathsf{ed}(\mathsf{exp}(G_i), \mathsf{exp}(G'_{t'+i-1}))$, *and* $\mathsf{ed}(\mathsf{exp}(G_{t+1})[..r], \mathsf{exp}(G'_{s'}))$.

Bhattacharya and Koucký [4] proved a similar result in the case where V' and V are appended to X and Y, respectively, from the left. By construction, the BK-decomposition is (almost) symmetric, which allows to adapt their argument to show an analogous result for the case where U and U' are appended to X and Y, respectively, from the right. Corollary 2.7 follows by first applying the result of Bhattacharya and Koucký [4] with $V' = \varepsilon$, and then using our analogous argument with $U' = \varepsilon$.

Finally, we use the following encoding of SLPs that allows reusing algorithms on strings for sequences of SLPs:

▶ **Fact 2.8** ([4, Lemma 3.13])**.** *Let $\mu = \tilde{O}(k)$ be a parameter. There is a mapping* enc *from the set of SLPs output by the BK-decomposition algorithm to the set of strings of length μ on an alphabet Γ of size polynomial in n (the maximum length of the input string of the BK-decomposition algorithm) that guarantees that the following is satisfied:*

1. *A grammar can be encoded and decoded in $O(\mu)$ time and space;*
2. *Encodings of two equal grammars are equal;*
3. *Encodings of two distinct grammars output by the decomposition algorithm differ in all μ characters with probability at least $1-1/n$.*

2.2.2 Proof of Theorem 2.3

A high-level idea of our algorithm is to apply the grammar decomposition to the stream, and then to reduce the problem to the problem of computing periods with mismatches via Corollary 2.7 and Fact 2.8. To this end, we need a stronger version of a streaming algorithm for computing periods with mismatches, and in particular, we introduce a *weight* function w on strings. In the case of the Hamming distance, this function is identically zero. In case of the edit distance, we define it as a partial function on Γ^*, namely, if $\Gamma^* \ni S = \mathrm{enc}(G_1 \cdots G_m)$, then $w(S) = |\mathsf{exp}(G_1 \cdots G_m)|$.

▶ **Proposition 2.9.** *The weight functions defined above satisfy each of the following:*

- *For a string S of length n, $w(S)$ can be computed in streaming deterministically using $t_w(n)$ time per character and $s_w(n)$ space;*
- *Consider strings X, Y. If at least two of $w(X), w(Y), w(XY)$ are defined, then the third one is defined as well and $w(XY) = w(X) + w(Y)$;*
- *Consider strings X, Y that have equal length in $[1..n]$. If $w(X), w(Y)$ are defined, then $w(Y)$ can be computed deterministically in $|\mathsf{MI}(X, Y)| \cdot t_w(n)$ time and $s_w(n)$ space.*

For the Hamming distance, $t_w(n) = O(1)$ and $s_w(n) = O(1)$ (trivially), and for the edit distance $t_w(n) = \tilde{O}(1)$ and $s_w(n) = \tilde{O}(k)$.

Proof. The claim is trivial for $w \equiv 0$. In the following, we consider the weight function for the edit distance. First, consider $S = \mathrm{enc}(G_1 G_2 \cdots G_m)$. Its weight $w(S)$ can be computed in streaming using $t_w(n)$ time per character, where $t_w(n) = \tilde{O}(1)$ and $s_w(n) = \tilde{O}(k)$ space thanks to Fact 2.8. The second property obviously holds. Finally, assume $X = \mathrm{enc}(G_1 G_2 \dots G_m)$ and $Y = \mathrm{enc}(G'_1 G'_2 \dots G'_m)$. Let $\mathcal{I} = \{i \le m, \mathrm{enc}(G_i) \neq \mathrm{enc}(G'_i)\}$. Given $\mathsf{MI}(X, Y)$, because of Fact 2.8, we have access to all characters in $\mathrm{enc}(G_i)$ and $\mathrm{enc}(G'_i)$ for $i \in \mathcal{I}$. As a result, we can compute $w(Y) = w(X) + \sum_{i \in \mathcal{I}} w(\mathrm{enc}(G'_i)) - w(\mathrm{enc}(G_i))$ in $\mathsf{MI}(X, Y) \cdot t_w(n)$ time and $s_w(n)$ space. ◀

In Section 3, we show the following theorem:

▶ **Theorem 2.10.** *Given a string T of length n and integers k, Δ, there is a streaming algorithm that computes all k-mismatch periods of T in $[1..n/2 + \Delta]$. Furthermore, for each detected k-mismatch period p, it also returns weight $w(T[p..])$ and $\mathsf{MI}(T[p..], T[..n - p + 1])$. The algorithm runs in $O(n \cdot k t_w(n) \log^5 n)$ time, uses $O(k^2 \log^3 n + s_w(n) + \Delta \cdot (k \log n + s_w(n)))$ space, and is correct w.h.p., where $t_w(n)$ and $s_w(n)$ are as defined in Proposition 2.9.*[7]

Let us now explain how it implies the algorithm for computing k-edit periods. Recall that we receive a string T of length n as a stream. In the first streaming pass on T, we apply Corollary 2.6. At every moment of the algorithm, we additionally store the number of definite grammars. By Corollary 2.6, we can then compute $m := |\mathcal{G}(T)|$ in $\tilde{O}(kn)$ time and $\tilde{O}(k)$ space.

In the second streaming pass, we again run the algorithm of Corollary 2.6. When a new character arrives, we update the SLPs and if a SLP G becomes definite, i.e. if we append G to the stream of definite grammars, we compute $\mathrm{enc}(G)$ in $\tilde{O}(k)$ time and space (Fact 2.8), and feed it character-by-character into the $\mu(k+2)$-mismatch period algorithm (Theorem 2.10), where μ is the parameter from Fact 2.8. Consider the moment when we arrive at the end of T and let $G_1, \dots, G_t$ be the active grammars. We compute $\mathrm{enc}(G_1 \cdots G_t)$ and feed it by character-by-character into the $\mu(k+2)$-mismatch period algorithm. Let $\mathcal{T}$ be the resulting stream we fed into the algorithm, i.e. $\mathcal{T} = \mathrm{enc}(\mathcal{G}(T))$, where $\mathcal{G}(T) = G_1^T G_2^T \cdots G_m^T$.

The $\mu(k+2)$-mismatch period algorithm retrieves all $\mu(k+2)$-mismatch periods of $\mathcal{T}$ one-by-one. When a $\mu(k+2)$-mismatch period p is retrieved, we do the following. If $p - 1$ is not a multiple of μ, we discard it immediately. Otherwise, by Corollary 2.7, there are at most $k + 2$ values $2 \le i \le m - (p-1)/\mu - 1$ such that $\mathrm{enc}(G_i^T) \neq \mathrm{enc}(G^T_{i+(p-1)/\mu})$. We retrieve the set I containing all such i from $\mathsf{MI}(\mathcal{T}[p..], \mathcal{T}[..m \cdot \mu - p + 1])$. We finally compute

$$d(p) = \sum_{i \in I} \mathsf{ed}(\mathsf{exp}(G_i^T), \mathsf{exp}(G^T_{i+(p-1)/\mu})) + \min_r \mathsf{ed}(\mathsf{exp}(G^T_{m-(p-1)/\mu})[..r], \mathsf{exp}(G_m^T))$$

[7] By taking $w \equiv 0$ and $\Delta = 0$, we immediately obtain Theorem 2.1.

If $d(p) \le k$, we compute all r' such that $\mathsf{ed}(\mathsf{exp}(G_1^T), \mathsf{exp}(G^T_{1+(p-1)/\mu}[r'..])) \le k - d(p)$. For all such r', we return $w(\mathrm{enc}(G^T_{(p-1)/\mu+2} \dots G^T_m)) + r'$ as a k-edit period of T.

Correctness. If $p \le n/2$ is a k-edit period of T, there exists i such that $\mathsf{ed}(T[..i], T[p..]) \le k$. By Corollary 2.7, there exist integers r, r', t, t' such that $T[..i] = \mathsf{exp}(G_1^T \cdots G_t^T) \cdot \mathsf{exp}(G^T_{t+1})[..r]$, $T[p..] = \mathsf{exp}(G_{t'})[r'..] \cdot \mathsf{exp}(G^T_{t'+1} \cdots G^T_m)$, $t + 1 = m - t' + 1$ and there are at most $k + 2$ indices i such that G_i^T and $G^T_{t'+i}$ mismatch. By Fact 2.8, $t' \cdot \mu + 1$ is a $\mu \cdot (k + 2)$-mismatch period of $\mathcal{T}$. Further, $t' \le (m + k)/2 + 1$:

▷ Claim 2.11. $t' \le (m + k)/2 + 1$.

Proof. We have $i \ge (n - p + 1) - k$ and hence $i + (n - p + 1) \ge n - k$. Consequently, $t+1+(m-t'+1) \ge m-k$. Indeed, assume towards a contradiction that $t+(m-t'+1) \le m-k$. We then have

$$
\begin{aligned}
&i + (n - p + 1) \le \\
&|\mathsf{exp}(G_1^T \cdots G_t^T) \cdot \mathsf{exp}(G^T_{t+1})[..r]| + |\mathsf{exp}(G_{t'})[r'..] \cdot \mathsf{exp}(G_{t'+1} \cdots G_m)| \le \\
&|\mathsf{exp}(G_1^T \cdots G^T_{t+1})| + |\mathsf{exp}(G^T_{t'} \cdots G^T_m)| \le |\mathsf{exp}(G_1^T \cdots G^T_{t+1})| + |\mathsf{exp}(G^T_{t+k+2} \cdots G^T_m)| \le \\
&n - k
\end{aligned}
$$

where the last inequality holds as each SLP G_i^T, $t + 1 < i < t + k + 2$ is non-trivial and hence its expansion has length at least one. Finally, since $t + 1 = m - t' + 1$, we obtain $m - t' + 1 \ge (m - k)/2$ and therefore $t' \le (m + k)/2 + 1$. ◁

Consequently, $t' \cdot \mu + 1$ is detected by the $\mu \cdot (k + 2)$-mismatch period algorithm with $\Delta = \mu(k/2 + 1) + 1$. We then identify p as a k-edit period of T by computing the edit distances between the expansions of the mismatching SLPs.

The algorithm fails if Corollary 2.7 fails, or if the $\mu \cdot (k + 2)$-mismatch period algorithm fails, or if Fact 2.8 fails, which happens with probability $\le \frac{2}{5}$ for n big enough. Also, notice that when Corollary 2.7 fails, the algorithm computes edit distance that is bigger than the actual distance. As a result, with a standard argument, we can run $\log n$ instances of the algorithm and return the minimal computed distance, to have the algorithm succeed w.h.p.

Complexity. The first pass on T takes $\tilde{O}(kn)$ time and $\tilde{O}(k)$ space. For the second pass, the $\mu \cdot (k + 2)$-mismatch period algorithm takes $\tilde{O}(\mu k t_w(n) n) = \tilde{O}(k^2 n)$ time and $\tilde{O}((\mu k)^2 + s_w(n)) = \tilde{O}(k^4)$ space by Theorem 2.10 and Proposition 2.9. For each $\mu \cdot (k + 2)$-mismatch period returned by the algorithm, we can bound the time needed to compute $d(p)$ as follows: Let $k_i = \mathsf{ed}(\mathsf{exp}(G_i), \mathsf{exp}(G_{i+(p-1)/\mu}))$. By Fact 2.4, we can compute k_i in time $\tilde{O}(k_i^2)$, hence if $d(p) \le k$, we can compute $d(p)$ in time $\tilde{O}(\sum_i k_i^2) = \tilde{O}(k^2)$. If the edit distance computations take total time more than $\tilde{O}(k^2)$, we can terminate them as we know that $d(p) > k$.[8] Theorem 2.3 follows.

3 Proof of Theorem 2.10

In this section, we prove Theorem 2.10. We first recall essential tools for the Hamming distance, then outline our algorithm and solve each case, and finally analyse it.

[8] To be more precise, we upper bound the number of longest common extension queries from Fact 2.4, it should not exceed $\sum_i (k_i + 1)^2 \le (k + 1)^2$.

3.1 Tools for the Hamming distance

For two strings P, T, a position $i \in [|P|..|T|]$ of T is a *k-mismatch occurrence* of P in T if $\mathsf{hd}(T[i-|P|+1..i], P) \leq k$. For an integer k, we define $\mathsf{hd}_{\leq k}(X,Y) = \mathsf{hd}(X,Y)$ if $\mathsf{hd}(X,Y) \leq k$ and ∞ otherwise. We denote by $Occ_k^H(P,T)$ the set of k-mismatch occurrences of P in T. For brevity, we define $\mathsf{hd}(P,T^*) = \mathsf{hd}(P,T^\infty[1..|P|])$.

▶ **Fact 3.1** ([9, Lemmas 6.2-6.4]). *Consider positive integers k, n, σ such that $k \leq n$ and $\sigma = n^{O(1)}$, and a family $\mathcal{U} \subseteq \{0,\ldots,\sigma-1\}^{\leq n}$ of strings. One can assign $O(k \log n)$-bit sketches $\mathsf{sk}_k(U)$ to all strings $U \in \mathcal{U}$ so that each of the following holds, assuming that all processed strings belong to $\mathcal{U}$:*

1. *Given sketches $\mathsf{sk}_k(U), \mathsf{sk}_k(V)$, there is an algorithm that uses $O(k \log^3 n)$ time to decide whether $\mathsf{hd}(U,V) \leq k$. If so, $\mathsf{MI}(U,V)$ is reported.*
2. *There is an algorithm that constructs one of $\mathsf{sk}_k(U), \mathsf{sk}_k(V)$ or $\mathsf{sk}_k(UV)$ given the two other sketches in $O(k \log n)$ time.*
3. *There is an algorithm that constructs $\mathsf{sk}_k(U)$ or $\mathsf{sk}_k(U^m)$ given the other sketch and the integer m in $O(k \log n)$ time.*
4. *If $\mathsf{hd}(U,V) = O(k)$, there is an algorithm that constructs $\mathsf{sk}_k(V)$ from $\mathsf{sk}_k(U)$ and $\mathsf{MI}(U,V)$ in $O(k \log^2 n)$ time.*

All algorithms use $O(k \log n)$ space and succeed w.h.p.

Below, we refer to the sketch of Fact 3.1 as the *k-mismatch sketch*. Note that for a string U and all integer $k' \geq k$, the sketch $\mathsf{sk}_{k'}(U)$ includes the sketch $\mathsf{sk}_k(U)$ (see [9] for a definition). Beyond the properties above, the k-mismatch sketch has one additional property:

▶ **Corollary 3.2** ([9, Fact 4.4]). *There exists a streaming algorithm that processes a string U in $O(k \log n)$ space using $O(\log^2 n)$ time per character so that the sketch $\mathsf{sk}_k(U)$ can be retrieved on demand in $O(k \log^2 n)$ time.*

By analysing the details of [9, Corollary 3.4], one can derive a streaming algorithm for computing all occurrences of a pattern in a text when the pattern is a prefix of some string and the text is a substring of the same string, which we refer to as the *k-mismatch algorithm*:

▶ **Corollary 3.3.** *Given a string T of length n, there is a streaming algorithm for a pattern $P = T[1..\ell]$ and a text $T[i..j]$, where $1 \leq i, j, \ell \leq n$, which uses $O(k \log^2 n)$ space and takes $O(k \log^4 n)$ time per arriving character. The algorithm reports all positions p such that $i+\ell-1 \leq p \leq j$ and $\mathsf{hd}(T[p-\ell+1..p], P) \leq k$ at the moment of their arrival. For each reported position p, $\mathsf{MI}(T[p-\ell+1..p], P)$ and $\mathsf{sk}_k(T[1..p-\ell])$ can be reported on demand in $O(k \log^2 n)$ time at the moment when p arrives. The algorithm is correct w.h.p.*

Proof. Clifford et al. [9, Corollary 3.4] presented an algorithm that reports the endpoints of all k-mismatch occurrences of a pattern in a text assuming that it first receives the pattern as a stream and then a text as a stream as well. The algorithm uses $O(k \log^2 n)$ space and takes $O(k \log^4 n)$ time per arriving character and is correct w.h.p.

This is not the current best algorithm (presented in Clifford et al. [9] as well). The reason why we selected the simpler algorithm is that it allows for the necessary preprocessing of the pattern to be easily done before one needs it for the text processing. Namely, the only information the algorithm stores about the pattern are the k-mismatch sketches of the prefixes of the pattern of the lengths equal to powers of two, and the k-mismatch sketch of the pattern itself, computed via Corollary 3.2, and the k-mismatch sketch of the pattern's prefix of length ℓ is never used before the position $\ell+1$ of the text. ◀

▶ **Fact 3.4** (cf. [8, Theorems 3.1 and 3.2]). *Given a pattern P of length m, a text T of length $n \leq \frac{3}{2}m$, and a threshold $k \in \{1, \ldots, m\}$, at least one of the following holds:*
1. *The number of k-mismatch occurrences of P in T is bounded by $576 \cdot n/m \cdot k$.*
2. *There is a primitive string Q of length $|Q| \leq m/128k$ that satisfies $\mathsf{hd}(P, Q^*) \leq 2k$.*

In the second case, the difference between the starting positions of any two k-mismatch occurrences of P in T is a multiple of $|Q|$ and if T' is the minimal substring of the text containing all k-mismatch occurrences of P in T, then $\mathsf{hd}(T', Q^*) \leq 6k$.

Let S be a string of length n. For our next lemma, we introduce the *forward cyclic rotation* $\mathsf{rot}(S) = S[n]S[1] \ldots S[n-1]$. In general, for $s \in \mathbb{N}$, a cyclic rotation $\mathsf{rot}^s(S)$ with shift s (resp. $-s$) is obtained by iterating rot (resp. rot^{-1}) s times. Note that a string S is primitive if and only if $\mathsf{rot}^s(S) = S$ implies $s = 0 \pmod{|S|}$.

▶ **Lemma 3.5.** *Given two strings X, Y such that X is a prefix of Y and $|Y| \leq \frac{5}{2}|X|$. Assume that there are primitive strings Q_X such that $\mathsf{hd}(Q_X^*, X) \leq k$ and $|Q_X| \leq \frac{|X|}{128k}$ and Q_Y such that $\mathsf{hd}(Q_Y^*, Y) \leq k$ and $|Q_Y| \leq \frac{|Y|}{128k}$. We then have $Q_X = Q_Y$.*

Proof. Suppose towards a contradiction that $Q_X \neq Q_Y$. By the triangle inequality, we have $\mathsf{hd}(Q_X^\infty[1..|X|], Q_Y^\infty[1..|X|]) \leq 2k$ and $\max\{|Q_X|, |Q_Y|\} \leq \frac{|Y|}{128k} \leq \frac{5|Y|}{256}$. Assume w.l.o.g. $|Q_X| \leq |Q_Y|$.

If there is $i \in \mathbb{N}$ such that $|Q_Y| = i|Q_X|$, then by primitivity of Q_Y, we have $\mathsf{hd}(Q_Y, Q_X^i) \geq 1$, and $\mathsf{hd}(Q_Y^\infty[1..|X|], Q_X^\infty[1..|X|]) \geq \frac{|X|}{|Q_Y|} \geq \frac{128k|X|}{|Y|} \geq \frac{256}{5}k > 2k$. Otherwise, for all $1 \leq j \leq |Q_X|$ we have

$$\begin{aligned}
&\mathsf{hd}(Q_Y^2, \mathsf{rot}^j(Q_X)^*) = \\
&= \mathsf{hd}(Q_Y, \mathsf{rot}^j(Q_X)^\infty[1..|Q_Y|]) + \mathsf{hd}(Q_Y, \mathsf{rot}^j(Q_X)^\infty[|Q_Y|+1..2|Q_Y|]) = \\
&= \mathsf{hd}(Q_Y, \mathsf{rot}^j(Q_X)^\infty[1..|Q_Y|)) + \mathsf{hd}(Q_Y, \mathsf{rot}^{j+|Q_Y|}(Q_X)^\infty[1..|Q_Y|]) \geq 1
\end{aligned}$$

The last inequality holds because $j \neq j + |Q_Y| \pmod{|Q_X|}$ and Q_X is primitive. Hence, $\mathsf{hd}(Q_Y^\infty[1..|X|], Q_X^\infty[1..|X|]) \geq \frac{|X|}{2|Q_Y|} \geq \frac{128}{5}k > 2k$, and $Q_Y = Q_X$. ◀

3.2 Structure of the algorithm

We can test a position in the following way to decide whether it is a k-mismatch period:

▶ **Proposition 3.6.** *Given $\mathsf{sk}_k(T)$, $\mathsf{sk}_k(T[1..p-1])$ and $\mathsf{sk}_k(T[1..n-p+1])$ for an integer $1 \leq p \leq n$, there is an algorithm that can decide whether p is a k-mismatch period of T using $O(k \log^3 n)$ time and $O(k \log n)$ space. In this case, it also returns $\mathsf{MI}(T[p..], T[..n-p+1])$. The algorithm is correct w.h.p.*

Proof. First, the algorithm applies Fact 3.1 to compute $\mathsf{sk}_k(T[p..])$ from $\mathsf{sk}_k(T)$ and $\mathsf{sk}_k(T[1..p-1])$ in $O(k \log n)$ time and space. By Observation 3.7, p is a k-mismatch period of T iff $h = \mathsf{hd}(T[1..n-p+1], T[p..n]) \leq k$. Given $\mathsf{sk}_k(T[p..])$ and $\mathsf{sk}_k(T[1..n-p+1])$, Fact 3.1 allows to decide whether $h \leq k$ in $O(k \log^3 n)$ time and $O(k \log n)$ space. ◀

For $p \in [n/2+1..n/2+\Delta]$, our algorithm computes the sketches required by Proposition 3.6 via Fact 3.1 and the weights $w(T[p..])$ via Proposition 2.9 using $O(nk \log n + \Delta t_w(n))$ total time and $O(\Delta(k \log n + s_w(n)))$ space. After reaching the end of T, the algorithm tests each of the candidates $p \in [n/2+1..n/2+\Delta]$ in $O(\Delta k \log^3 n) = O(nk \log^3 n)$ total time and $O(k \log n)$ space, and for each k-mismatch period, returns p, $w(T[p..])$, and $\mathsf{MI}(T[p..], T[..n-p+1])$.

The rest of the section is devoted to computing periods $p \in [1..n/2]$. The following simple observation is crucial for the correctness of our algorithm.

▶ **Observation 3.7.** *An integer $1 \le p \le n$ is a k-mismatch period of T iff* $\mathsf{hd}(T[1..n-p+1], T[p..n]) \le k$. *As a corollary, if p is a k-mismatch period of T, then for all $1 \le \ell \le n-p$, the position p is the starting position of a k-mismatch occurrence of $T[1..\ell+1]$.*

It follows that we can use k-mismatch occurrences of appropriately chosen prefixes of T in T to filter out candidate k-mismatch periods. For $j = 1, \ldots, \lceil \log_{3/2} n \rceil$, define $\ell_j := \lfloor n/(3/2)^j \rfloor$, $P_j = T[1..\ell_j]$, and $T_j = T[\max\{\lfloor n/2 \rfloor - \ell_j, 1\}..\lfloor n/2 \rfloor - \ell_{j+1} + \ell_j - 2]$. For each j, we give two algorithms run in parallel, which together compute the set $\mathcal{P}_k^j$ of all k-mismatch periods of T in the interval $[\lfloor n/2 \rfloor - \ell_j..\lfloor n/2 \rfloor - \ell_{j+1} - 1]$. The first algorithm assumes that the number of k-mismatch occurrences of P_j in T_j is at most $K = 576k$ (we call such P_j "non-periodic", slightly abusing the standard definition), while the second one is correct when the number of occurrences is larger than K (we call such P_j "periodic").

3.3 The algorithm for non-periodic P_j

We maintain the sketch and the weight of the current text T using Corollary 3.2 and Proposition 2.9 in $O(\log^2 n + t_w(n))$ time per character and $O(k \log n + s_w(n))$ space. After reading $P_j = T[1..\ell_j]$, we memorise $w(P_j)$. Additionally, we run the k-mismatch algorithm (Corollary 3.3) for a pattern P_j and a text T_j, which in particular computes $\mathsf{sk}_k(P_j)$. Furthermore, we maintain two hash tables, each of size at most K, Pref_j and Suf_j. Intuitively, we want Pref_j to contain every position p which is the starting position of a k-mismatch occurrence of P_j in T_j, associated with $\mathsf{sk}_k(T[1..p-1])$ and the weight of $T[1..p-1]$. As for the table Suf_j, we would like it to contain every position t such that $n-t+1 \in \mathrm{Pref}_j$, again associated with $\mathsf{sk}_k(T[1..t])$ and the weight of $T[1..t]$. We implement Pref_j and Suf_j via the cuckoo hashing scheme [22] and de-amortise as explained in [2, Theorem A.1] to yield the following:

▶ **Fact 3.8** ([22, 2])**.** *A set of K integers in $\{0,1\}^w$, where $w = \Theta(\log n)$ is the size of the machine word, can be stored in $O(K \log n)$ space while maintaining look-up queries in $O(1)$ worst-case time and insertions in $O(1)$ worst-case time w.h.p.*

When we receive a character $T[p]$, the tables are updated as follows. Assume first that the k-mismatch algorithm detects a new occurrence of P_j ending at the position p. We retrieve $\mathsf{sk}_k(T[1..p-\ell_j])$ and $\mathsf{MI}(P_j, T[p-\ell_j+1..p])$ in $O(k \log^2 n)$ time (Fact 3.1). Furthermore, with $w(P_j)$, we can deduce $w(T[p-\ell_j+1..p])$, and finally, with $w(T[..p])$, we compute $w(T[1..p-\ell_j])$, and add $p-\ell_j$ associated with $\mathsf{sk}_k(T[1..p-\ell_j])$ and $w(T[1..p-\ell_j])$ to Pref_j in $O(kt_w(n))$ time and $O(s_w(n))$ space. Secondly, if for $t = n-p$ we have $t \in \mathrm{Pref}_j$, we add p associated with $\mathsf{sk}_k(T[1..p])$ to Suf_j. If either of the two insertions takes more than constant time or if the size of any of Pref_j and Suf_j becomes larger than K, the algorithm terminates and returns $\perp$. Assume that the algorithm has reached the end of T.

▶ **Proposition 3.9.** *If $t \in \mathcal{P}_k^j$, then $t-1 \in \mathrm{Pref}_j$ and $n-t+1 \in \mathrm{Suf}_j$.*

Proof. As $t \in \mathcal{P}_k^j$, by Observation 3.7 $t + \ell_j - 1$ is the ending position of a k-mismatch occurrence of P_j in T. Furthermore, $t \in [\lfloor n/2 \rfloor - \ell_j..\lfloor n/2 \rfloor - \ell_{j+1} - 1]$, and hence $T[t, t+\ell_j-1]$ is a fragment of T_j. This proves that $t-1 \in \mathrm{Pref}_j$. To show that $n-t+1 \in \mathrm{Suf}_j$, note that

$$2t \le 2(\lfloor n/2 \rfloor - \ell_{j+1} - 1) \le n - 2\ell_{j+1} - 2 \le n - \ell_j.$$

Therefore, $t + \ell_j - 1 < n - t + 1$, and $t-1$ will be added to Pref_j before $n-t+1$. The claim follows. ◀

Finally, the algorithm considers each position $t \in \mathrm{Pref}_j$, extracts $\mathsf{sk}_k(T[1..t])$ from Pref_j and $\mathsf{sk}_k(T[1..n-t])$ from Suf_j and if t passes the test of Proposition 3.6, reports $t+1$ as a k-mismatch period of T, and returns $w(T[t+1..]) = w(T) - w(T[1..t])$.

▶ **Proposition 3.10.** *Assume that the number of occurrences of P_j in T_j is at most $K = 576k$. The algorithm computes $\mathcal{P}_k^j$ and $\mathsf{MI}(T[p..], T[..n-p+1]), p \in \mathcal{P}_k^j$ in $O(n(k\log^4 n + t_w(n)))$ time and $O(k^2\log^2 n + s_w(n))$ space and is correct w.h.p.*

Proof. The algorithm runs an instance of the k-mismatch algorithm (Corollary 3.3) that takes $O(nk\log^4 n)$ time and $O(k\log^2 n)$ space. Computing weights takes $O(nt_w(n))$ time and $s_w(n)$ space. Adding elements to Pref_j and Suf_j, as well as look-ups, takes $O(1)$ time per element, and we add at most $K = O(k)$ elements in total. The two hash tables occupy $O(k^2\log^2 n)$ space (Fact 3.8, Fact 3.1). Finally, testing all candidate positions requires $O(K \cdot k\log^3 n) = O(nk\log^3 n)$ time and $O(k\log n)$ space (Proposition 3.6). The correctness of the algorithm follows from Proposition 3.9 and Proposition 3.6. The algorithm can fail if the k-mismatch algorithm errs, or if adding an element to the hash tables takes more than constant time, or if the test fails. By the union bound, Corollary 3.3, Fact 3.1, and Proposition 3.6, the failure probability is inverse-polynomial in n. ◀

3.4 The algorithm for periodic P_j

We first explain how we preprocess P_j. Recall the Boyer–Moore majority vote algorithm:

▶ **Fact 3.11** ([7])**.** *Given a sequence $e_1, \ldots, e_m$ of elements, there is a streaming algorithm that stores $O(1)$ elements of the sequence and returns a majority element if there exists one (otherwise, it can return an arbitrary element). Assuming constant-time access and comparison on the elements, the algorithm takes $O(m)$ time.*

▶ **Lemma 3.12.** *Given a prefix $P = T[1..\ell]$ of T, there is a streaming algorithm that uses $O(k(s_w(n) + \log^2 n))$ space and runs in $O(\ell k\log^4 n + \ell \cdot t_w(n) + k^2\log^2 n)$ time. If there is a primitive string Q of length $|Q| \le \frac{\ell}{128k}$ that satisfies $\mathsf{hd}(P, Q^*) < 2k$, the algorithm computes, correctly w.h.p., $|Q|$, $\mathsf{sk}_{3k}(Q)$, and $w(Q)$ before or upon the arrival of $T[(\lfloor \ell/|Q| \rfloor - 2) \cdot |Q|]$. If there is no such string, the algorithm determines it before $T[\ell]$ arrives.*

Proof. The main idea of the lemma is that $|Q|$ must be the starting position of the first $\Theta(k)$-mismatch occurrence of $P[1..\lfloor \frac{\ell}{2} \rfloor]$ in $T[2..]$. As soon as we know $|Q|$, we compute the $3k$-mismatch sketches and the weights of $\Theta(k)$ consecutive substrings of length $|Q|$. By Fact 3.4, the majority of them equal Q, which allows computing $\mathsf{sk}_{3k}(Q)$ and $w(Q)$ via the Boyer–Moore majority vote algorithm [7]. We now provide full details.

For brevity, let $\ell' = \lfloor \frac{\ell}{2} \rfloor$. We run the $8k$-mismatch algorithm for a pattern $P[1..\ell']$ and a text $T[2..]$. If the $8k$-mismatch algorithm does not detect an occurrence of P before or when reading the position $\ell' + \frac{\ell}{128k} < \ell$, the algorithm concludes that Q does not exist and terminates. Assume now that the algorithm does detect a $8k$-mismatch occurrence of the pattern ending at a position p, $p \le \ell' + \frac{\ell}{128k}$, and let it be the first detected occurrence. The instance of the $8k$-mismatch algorithm is immediately terminated and we launch the majority vote algorithm (Fact 3.11). Let $q = p - \ell' + 1$ and p' the smallest multiple of q greater than p. We then compute $\mathsf{sk}_{3k}(T[p'+1..p'+q])$, $\mathsf{sk}_{3k}(T[p'+q+1..p'+2q])$, $\ldots$, $\mathsf{sk}_{3k}(T[p'+(12k-1)q+1..p'+12kq])$ via Corollary 3.2 and feed them into the majority vote algorithm. After all the $12k$ sketches have been computed, which happens when we read the position $p' + 12kq \le (\lfloor \ell/q \rfloor - 2) \cdot q$, we return q and the output of the majority vote algorithm as $\mathsf{sk}_{3k}(Q)$. Using the same majority vote approach, we compute $w(Q)$.

We now show the correctness of the algorithm. We start by showing that if Q exists, then $q = |Q|$. Let i be a multiple of $|Q|$ smaller than $\ell/2$. By Fact 3.4 and the triangle inequality, we have $\mathsf{hd}(P[1..\ell'], T[i+1..i+\ell']) \leq \mathsf{hd}(P[1..\ell'], Q^*) + \mathsf{hd}(Q^*, T[i+1..i+\ell']) \leq 8k$. Reciprocally, if i is not a multiple of $|Q|$, then by primitivity of Q we have $\mathsf{hd}(Q^\infty[1..\ell'], Q^\infty[i+1..i+\ell']) \geq \lfloor \ell'/|Q| \rfloor \geq 62k$. Furthermore, by the triangle inequality, $\mathsf{hd}(Q^\infty[1..\ell'], Q^\infty[i+1..i+\ell']) \leq 8k + \mathsf{hd}(P[1..\ell'], T[i+1..i+\ell'])$, which implies $\mathsf{hd}(P[1..\ell'], T[i+1..i+\ell']) \geq 54k$. Consequently, if Q exists, at least $6k$ of the strings $T[p'+1..p'+q]$, $T[p'+q+1..p'+2q]$, $\ldots$, $T[p'+(12k-1)q+1..p'+12kq]$ are equal to Q, and the majority vote algorithm indeed outputs $sk_{3k}(Q)$ (assuming that neither the $8k$-mismatch algorithm nor the algorithm of Corollary 3.2 did not err, which is true w.h.p.).

We finally analyse the complexity of the preprocessing step. The $8k$-mismatch pattern matching algorithm (Corollary 3.3) takes $O(k \log^4 n)$ time per character and $O(k \log^2 n)$ space. The algorithm of Corollary 3.2 uses $O(\log^2 n)$ time per character $O(k \log n)$ space. Furthermore, $O(k)$ sketches are retrieved, which takes $O(k^2 \log^2 n)$ time (Corollary 3.2) and maintaining weights requires $t_w(n)$ time and $k \cdot s_w(n)$ space. Finally, the majority vote algorithm takes $O(k \log n)$ space and $O(k \log n)$ time. In total, the algorithm takes $O(k(s_w(n) + \log^2 n))$ space and $O(\ell k \log^4 n + \ell \cdot t_w(n) + k^2 \log^2 n)$ time. ◀

We now apply the lemma above to preprocess P_j as follows. We maintain the $3k$-mismatch sketch of T using Corollary 3.2. Fact 3.4 ensures that if there are at least K occurrences of P_j in T_j, then there exists a primitive string Q of length $q := |Q| \leq \frac{\ell_j}{128k}$ such that $\mathsf{hd}(Q^*, P_j) < 2k$. For brevity, let $\lambda_j = (\lfloor \ell_j/q \rfloor - 2) \cdot q$. We apply Lemma 3.12 to P_j and condition on the fact that it outputs q, $\mathsf{sk}_{3k}(Q)$, and $w(Q)$ before or upon the arrival of $T[\lambda_j]$. By an application of Fact 3.1, we compute $\mathsf{sk}_{3k}(Q^{\lfloor \lambda_j/q \rfloor + 1})$ and then $\mathsf{MI}(P_j[..\lambda_j+q], Q^{\lfloor \ell_j/q \rfloor + 1})$ using $\mathsf{sk}_{3k}(P_j[..\lambda_j+q])$. To finish the preprocessing, we compute one more sketch:

▶ **Lemma 3.13.** *Assume there are $\geq K$ k-mismatch occurrences of P_j in T_j. There is a streaming algorithm that uses $O(k \log^2 n + s_w(n))$ space and $O(k \log^4 n + t_w(n))$ time per character, and computes $\mathsf{sk}_k(Q[..r])$ for $r := n - 2p \pmod q$ and $w(Q[..r])$ correctly w.h.p. upon arrival of $T[p]$, where p is the endpoint of the first k-mismatch occurrence of P_j in T_j.*

Proof. By the condition of the lemma, Q is defined and we assume to have computed it by arrival of $T[\lambda_j]$, where $\lambda_j = (\lfloor \ell_j/q \rfloor - 2) \cdot q$ and $q = |Q|$. We run two instances of the k-mismatch algorithm: One for a pattern $P_j[..\lambda_j]$ and T_j, and the other for P_j and T_j. Assume that we detect a k-mismatch occurrence of $P_j[..\lambda_j]$ ending at a position x. Let $r' = n - 2(x + \ell_j - \lambda_j) \pmod q$. We compute $\mathsf{sk}_k(T[x+1..x+r'])$ via Corollary 3.2, and $w(T[x+1..x+r'])$ via Proposition 2.9. If $p = x + \ell_j - \lambda_j$ is not the ending position of a k-mismatch occurrence of P_j, we discard the computed information and continue. Otherwise, we have $r' = r$. At the position p the k-mismatch algorithm extracts $\mathsf{MI}(T[p - \ell_j + 1..p], P_j)$ in $O(k)$ time, and we use it to extract $\mathsf{MI}(T[x+1..x+r], Q[..r])$ from $\mathsf{MI}(P_j[..\lambda_j + q], Q^*)$ in $O(k)$ time as well. Note that the size of the extracted mismatch information is at most $6k$ by Fact 3.4. Finally, we apply Fact 3.1 to compute $\mathsf{sk}_k(Q[..r])$ from $\mathsf{sk}_k(T[x+1..x+r])$ and $\mathsf{MI}(T[x+1..x+r], Q[..r])$ in $O(k \log^2 n)$ time and $O(k \log n)$ space. Similarly, with $w(T[x+1..x+r])$ and the mismatch information, we compute $w(Q[..r])$ in $O(kt_w(n))$ time and $O(s_w(n))$ space. (See Figure 2 for an illustration.)

To show the complexity of the algorithm, we need to understand the structure of the k-mismatch occurrences of $P_j[..\lambda_j]$ in T_j. As each k-mismatch occurrence of P_j starts with a k-mismatch occurrence of $P_j[..\lambda_j]$, T_j contains at least K k-mismatch occurrences of the latter. Since $|T_j| \leq 2\ell_j - \ell_{j-1} \leq \frac{3}{2}\lambda_j$, by Fact 3.4 there is a primitive string Q' such that $|Q'| \leq \frac{\lambda_j}{128k}$ and $\mathsf{hd}(P_j[..\lambda_j], (Q')^*) \leq 2k$. By Lemma 3.5, $Q = Q'$, and again by Fact 3.4,

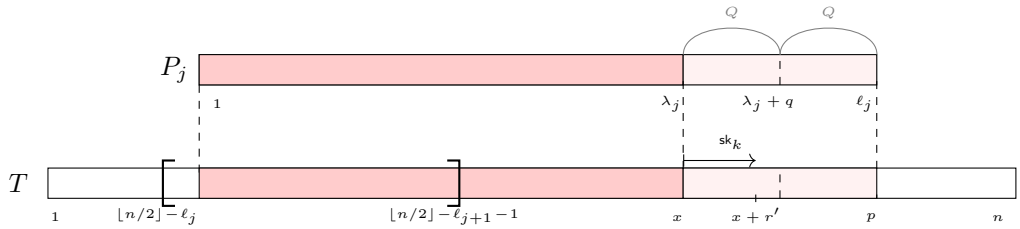


Figure 2 When we detect a k-mismatch occurrence of $P_j[..\lambda_j]$, we use the next q characters to compute the candidate for $\mathsf{sk}_k(Q[..r])$. If the k-mismatch occurrence of $P_j[..\lambda_j]$ extends to a k-mismatch occurrence of P_j, then we keep the candidate sketch.

the difference between the starting positions of any two k-mismatch occurrences of $P_j[1..\lambda_j]$ in T_j is a multiple of q. Therefore, we never process more than two k-mismatch occurrences of $P_j[..\lambda_j]$ at a time and the bounds follow. ◀

This information will allow us computing the sketches necessary for Proposition 3.6.

3.4.1 Main phase of the algorithm

The main phase distinguishes two cases: $j = 1, 2$ and $j \geq 3$. For $j = 1, 2$, we show the following result:

▶ **Proposition 3.14.** *Assume that $j = \{1, 2\}$ and that P_j has more than $K = 576k$ k-mismatch occurrences in T_j. The algorithm computes $\mathcal{P}_k^j$, $\mathsf{MI}(T[p..], T[..n - p + 1])$ and $w(T[p..])$ for $p \in \mathcal{P}_k^j$ in $O(n \cdot kt_w(n) \log^4 n)$ time and $O(k^2 \log^2 n + s_w(n))$ space and is correct w.h.p.*

Proposition 3.14 can be proven using the same ideas as in the case $j \geq 3$, but needs some additional care because the size of the pattern being large ($n/4$ or $n/2$) leads to edge cases that are treated separately. Below, we focus on the case $j \geq 3$. We start by extending P_j into a prefix P'_j (Algorithm 1).

Algorithm 1 Extension of P_j into P'_j of length ℓ'_j: case $j \geq 3$.

```
ℓ'_j ← λ_j, sk^1 ← sk_{3k}(T[1..λ_j]), w_1 ← w(T[1..λ_j]);
while ℓ'_j < 2ℓ_j do
    sk^2 ← sk^1, w_2 ← w_1;
    ℓ'_j ← ℓ'_j + q, sk^1 ← sk_{3k}(T[1..ℓ'_j]), w_1 ← w(T[1..ℓ'_j]);
    if h = MI(T[1..ℓ'_j], Q*) > 2k then
        return (ℓ'_j, sk^1, sk^2, w_1, w_2);          // Use sk_{3k}(T) and sk_{3k}(Q)
return (ℓ'_j, sk^1, sk^2, w_1, w_2);
```

The following inequalities are essential for analysis of correctness of the algorithm:

▶ **Proposition 3.15.** *For $j \geq 3$, we have $\ell'_j < n$ and $\lfloor n/2 \rfloor - \ell_{j+1} - 1 + (\ell'_j - 1) \leq n$.*

Proof. We start by showing the first inequality:

$$\ell'_j < 2\ell_j + q \leq \ell_j \cdot (2 + 1/128k) \leq (8n/27) \cdot (2 + 1/128k) \leq n$$

To show the second inequality, note that

$$\lfloor n/2 \rfloor - \ell_{j+1} - 1 + \ell'_j - 1 \leq$$
$$\lfloor n/2 \rfloor - n/(3/2)^{j+1} + 2n/(3/2)^j + q \leq \lfloor n/2 \rfloor + (4/3 + 1/128) \cdot n/(3/2)^j \leq n \quad \blacktriangleleft$$

We now discuss how to implement Algorithm 1. As we know $\mathsf{sk}_{3k}(Q)$ before or upon the arrival of $T[\lambda_j]$, Algorithm 1 can be implemented in streaming via Corollary 3.2, Fact 3.1 to use $O(k \log n + s_w(n))$ space and $O(k \log^2 n + t_w(n))$ time per character. Furthermore, it always terminates before the arrival of $T[n]$ (Proposition 3.15) and outputs $\ell'_j := |P'_j|$, $\mathsf{sk}_{3k}(P'_j)$ and $w(P'_j)$, and $\mathsf{sk}_{3k}(P'_j[1..\ell'_j - q])$ and $w(P'_j[1..\ell'_j - q])$. Define $T'_j = T[\max\{\lfloor n/2 \rfloor - \ell_j, 1\}..\lfloor n/2 \rfloor - \ell_{j+1} + \ell'_j - 2]$. Note that T'_j is well-defined by Proposition 3.15. Furthermore, let $P''_j = [1..\ell'_j - q]$, $\ell''_j = |P''_j|$, and $T''_j = T[\lfloor n/2 \rfloor - \ell_j..\lfloor n/2 \rfloor - \ell_{j+1} + \ell''_j - 2]$. From Observation 3.7 and Proposition 3.15 we obtain:

▶ **Corollary 3.16.** *The set $\mathcal{P}^j_k$ is a subset of the set of the starting positions of k-mismatch occurrences of P'_j in T'_j, and consequently of the set of the starting positions of k-mismatch occurrences of P''_j in T''_j.*

Consider now two subcases depending on the line where Algorithm 1 executes the return.

3.4.1.1 Return is executed in Line 6

▶ **Proposition 3.17.** *The prefixes P'_j and P''_j have the following properties:*
1. *The number of k-mismatch occurrences of P'_j in T'_j is $O(k)$.*
2. *The distance between any two k-mismatch occurrences of P''_j in T''_j is a multiple of q.*

Proof. To show the first part of the claim, note that $\ell_j \leq \ell'_j \leq \frac{5}{2}\ell_j$ and $|T'_j| \leq \ell_j - \ell_{j+1} + \ell'_j \leq \frac{3}{2}\ell'_j$. Assume, for sake of contradiction, that there are more than $576k$ occurrences of P'_j in T'_j. By Fact 3.4, there is a primitive string Q', $|Q'| \leq \ell'_j/128k$, such that $\mathsf{hd}(P'_j, (Q')^*) \leq 2k$. By Lemma 3.5, $Q' = Q$, a contradiction.

We now show the second part of the claim. Note that $\frac{2}{3}\ell_j \leq \ell_j - q \leq \ell''_j \leq 2\ell_j$ and hence $|T''_j| \leq \ell_j - \ell_{j+1} + \ell''_j \leq \frac{3}{2}\ell''_j$. Next, we have two cases: $\ell''_j \leq \ell_j$ and $\ell''_j > \ell_j$. In the first case, every position p which is a starting position of a k-mismatch occurrence of P_j in T_j is also a starting position of a k-mismatch occurrence of P''_j in T''_j. Hence, there are at least $K = 576k$ k-mismatch occurrences of P''_j in T''_j, and by Fact 3.4, there is a primitive string Q'', $|Q''| \leq \ell''_j/128k$, such that $\mathsf{hd}(P''_j, (Q'')^*) \leq 2k$. By Lemma 3.5, $Q'' = Q$. If $\ell''_j > \ell_j$, then $\mathsf{hd}(P''_j, Q^*) \leq 2k$ by construction. Consequently, by applying Fact 3.4 one more time, we obtain that the difference between the starting positions of any two k-mismatch occurrences of P''_j in T''_j is q. ◀

We apply the k-mismatch algorithm to detect k-mismatch occurrences of P''_j and P'_j in the text T'_j. In parallel, we maintain two hash tables of size at most $K = 576k$ each, Pref_j and Suf_j, implemented via Fact 3.8. When we receive a character $T[p]$, the tables are updated as follows. Assume first that the k-mismatch algorithm detects a new occurrence of P''_j ending at the position p. We retrieve $\mathsf{sk}_{3k}(T[1..p - \ell''_j])$ and $\mathsf{MI}(P''_j, T[p - \ell''_j + 1..p])$. From the mismatch information, $w(P''_j)$ and $w(T[..p])$ we compute $w(T[1..p - \ell''_j])$, and we memorise $t = p - \ell''_j$ associated with the sketch and the weight for the next q positions. Importantly, at every moment the algorithm stores at most one position-sketch pair by Proposition 3.17. By the definition of P''_j and T''_j, the k-mismatch occurrences detected by the algorithm can only start before $\lfloor n/2 \rfloor$, and consequently $2(t+1) \leq n$, which implies $t + 1 < n - t$.

Now, assume that $t+1 < n-t \leq p$. In this case, we immediately compute $\mathsf{sk}_k(T[1..n-t])$ via the following claim and memorise it for the next q positions:

▶ **Proposition 3.18.** *Assume to be given* $\mathsf{sk}_k(T[1..t])$, $\mathsf{sk}_k(Q[r..])$, *and* $\mathsf{sk}_k(Q)$. *There is an algorithm that uses* $O(k \log n)$ *space and* $O(k \log^3 n)$ *time and computes* $\mathsf{sk}_k(T[1..n-t])$. *The algorithm succeeds w.h.p.*

Proof. By Corollary 3.16 and Proposition 3.17, $(n-t) - t = iq + r$ for some integer i. Consequently, the sketch can be computed as follows. First, the algorithm computes $\mathsf{MI}(P''_j, T[p - \ell''_j + 1..p])$ and $\mathsf{MI}(P''_j, Q^*)$. Secondly, it computes $\mathsf{sk}_k(Q^i Q[1..r])$ and then deduces $\mathsf{sk}_k(T[t+1..n-t])$ in $O(k \log n)$ space and $O(k \log^3 n)$ time via Fact 3.1. Finally, it computes $\mathsf{sk}_k(T[1..n-t])$ from $\mathsf{sk}_k(T[1..t])$ and $\mathsf{sk}_k(T[t+1..n-t])$. ◀

Also, if the current position p equals $n-t$, where t is the position in the stored position-sketch pair, we memorise $\mathsf{sk}_k(T[1..n-t])$. Again, by Proposition 3.17 the algorithm stores at most one sketch at a time. If the current position p is the endpoint of a k-mismatch occurrence of P'_j in T'_j, the position $p-q$ is necessarily the endpoint of a k-mismatch occurrence of P''_j in T''_j, and we store $t = p - q - \ell''_j$ associated with $\mathsf{sk}_k(T[1..t])$ and $w(T[1..t])$. We add this triple to Pref_j. In addition, if $n - t \leq p$, we have already computed $\mathsf{sk}_k(T[1..n-t])$, and we add it to Suf_j. Finally, if p is the current position and for $t = n - p$ we have $(t, \mathsf{sk}_k(T[1..t]), w(T[1..t])) \in \mathrm{Pref}_j$, we add p associated with $\mathsf{sk}_k(T[1..p])$ to Suf_j.

If any of the insertions takes more than constant time or if the size of any of Pref_j and Suf_j becomes larger than K, the algorithm terminates and returns $\perp$.

When the entire string T has arrived, the algorithm considers each position $t \in \mathrm{Pref}_j$, extracts $\mathsf{sk}_k(T[1..t]), w(T[1..t])$ from Pref_j and $\mathsf{sk}_k(T[1..n-t])$ from Suf_j and if t passes the test of Proposition 3.6, reports $t+1$ as a k-mismatch period of T, and also returns $w(T[t+1..]) = w(T) - w(T[1..t])$ (undefined if one of the values on the right is undefined), and $\mathsf{MI}(T[t+1..], T[..n-t])$.

3.4.1.2 Return is executed in Line 7

▶ **Proposition 3.19.** *If return is executed in Line 7, we have* $\mathsf{hd}(P'_j, Q^*) < 2k$ *and for all* $t \in [\lfloor n/2 \rfloor - \ell_j .. \lfloor n/2 \rfloor - \ell_{j+1} - 1]$ *there is* $\lfloor n/2 \rfloor \leq n - t \leq \lfloor n/2 \rfloor - \ell_j + (\ell'_j + 2)$.

Proof. The first part of the claim is immediate by construction. To show the second part, recall that $\ell'_j \geq 2\ell_j$. Hence,

$$\lfloor n/2 \rfloor \leq n - t \leq n - \lfloor n/2 \rfloor + \ell_j + 1 \leq \lfloor n/2 \rfloor - \ell_j + (2\ell_j + 2) \leq \lfloor n/2 \rfloor - \ell_j + (\ell'_j + 2)$$ ◀

We run the k-mismatch algorithm (Corollary 3.3) for P'_j and T'_j. If a position p is the endpoint of the first k-mismatch occurrence of P'_j in T'_j, we retrieve $\mathsf{sk}_k(T[1..p - \ell'_j])$ and $\mathsf{MI}(T[p - \ell'_j + 1..p], P'_j)$, and deduce $w(T[1..p - \ell'_j])$ with the same method as in the previous case. Also, we memorise p, the sketch, the mismatch information and the weight. We would now like to process the last k-mismatch occurrence of P'_j in T'_j in a similar way. As it is not possible to say in advance whether the current k-mismatch occurrence is the last one, we instead do the following. Starting from the second endpoint p' of a k-mismatch occurrence of P'_j in T'_j, we retrieve $\mathsf{MI}(T[p' - \ell'_j + 1..p'], P'_j)$ by Corollary 3.3, and memorise p' and the mismatch information until the next k-mismatch occurrence is detected, when they are discarded. Additionally, at the position $p'+1$ we launch a new instance of the algorithm of Corollary 3.2, maintaining $\mathsf{sk}_k(T[p'+1..])$. Again, if we detect another k-mismatch occurrence, we discard the currently stored sketch. In addition, when $T[x]$ arrives, for

$x \in \{\lfloor n/2 \rfloor - \ell_j + (\ell'_j + 1), \lfloor n/2 \rfloor - \ell_j + (\ell'_j + 2)\}$ we launch a new instance of the algorithm in Corollary 3.2, maintaining $\mathsf{sk}_k(T[x+1..])$. This way, when we reach the end of T, we have the following information at hand:

1. For the endpoint p of the first k-mismatch occurrence of P'_j in T_j, $\mathsf{sk}_k(T[1..p-\ell'_j])$, weight $w(T[1..p-\ell'_j])$, and the mismatch information $\mathsf{MI}(T[p-\ell'_j+1..p], P'_j)$;
2. For the endpoint p' of the last k-mismatch occurrence of P'_j in T_j, the mismatch information $\mathsf{MI}(T[p'-\ell'_j+1..p'], P'_j)$ and $\mathsf{sk}_k(T[p'+1..])$;
3. $\mathsf{sk}_k(T[x+1..])$ for $x \in \{\lfloor n/2 \rfloor - \ell_j + (2\ell_j + 1), \lfloor n/2 \rfloor - \ell_j + (2\ell_j + 2)\}$.

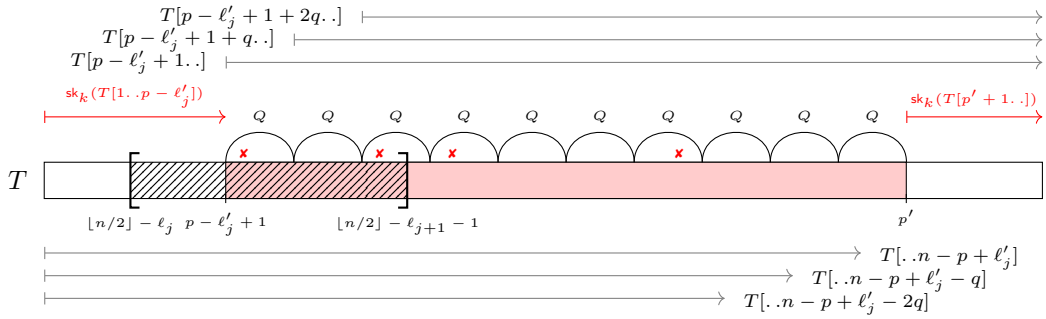


Figure 3 If Algorithm 1 executes return in Line 7, $\mathcal{P}^k_j \subseteq \{p-\ell'_j+1, p+q-\ell'_j+1, \ldots, p'-\ell'_j+1\}$. To test each position in the latter set, we exploit the structure of $T[p-\ell'_j+1..p'-\ell'_j+1]$ (shown in red, with crosses marking mismatches between it and Q^∞).

By Proposition 3.19 and Fact 3.4, we have $\mathcal{P}^k_j \subseteq \{p-\ell'_j+1, p+q-\ell'_j+1, \ldots, p'-\ell'_j+1\}$. We test each position t in the latter set as follows (see Figure 3):

1. **Compute $\mathsf{sk}_k(T[t..])$.** First, retrieve $\mathsf{sk}_k(T[1..t-1])$ from sketches $\mathsf{sk}_k(T[1..p-\ell'_j])$ and $\mathsf{sk}_k(Q^{(t-p)/q})$, and the mismatch information $\mathsf{MI}(T[p-\ell'_j+1..p], P'_j)$, $\mathsf{MI}(T[p'-\ell'_j+1..p'], P'_j)$, and $\mathsf{MI}(P'_j, Q^*)$ in $O(k \log^2 n)$ time and $O(k \log n)$ space via Fact 3.1. Second, compute $\mathsf{sk}_k(T[t..])$ from $\mathsf{sk}_k(T)$ and $\mathsf{sk}_k(T[1..t-1])$ in $O(k \log^2 n)$ time and $O(k \log n)$ space via another application of Fact 3.1.
2. **Compute $w(T[t..])$.** First, retrieve $w(T[1..t-1])$ from $w(T[1..p-\ell'_j])$ and $w(Q^{(t-p)/q}) = [(t-p)/q] \cdot w(Q)$, and the mismatch information $\mathsf{MI}(T[p-\ell'_j+1..p], P'_j)$, $\mathsf{MI}(T[p'-\ell'_j+1..p'], P'_j)$, and $\mathsf{MI}(P'_j, Q^*)$ in $O(kt_w(n))$ time and $s_w(n)$ space. Second, compute $w(T[t..]) = w(T) - w(T[1..t-1])$ (undefined if one of the values on the right is undefined).
3. **Compute $\mathsf{sk}_k(T[1..n-t+1])$.** If $n-t+1 \in \{\lfloor n/2 \rfloor - \ell_j + (\ell_j + 1), \lfloor n/2 \rfloor - \ell_j + (\ell_j + 2)\}$, we already know the sketch. Otherwise, by Proposition 3.19 $\lfloor n/2 \rfloor \le n-t+1 \le \lfloor n/2 \rfloor + \ell'_j$. Additionally, $(n-t+1) - (p-\ell'_j+1) + 1 = q \cdot i + r$ for an integer r defined as in Lemma 3.13. Hence, $\mathsf{sk}_k(T[1..n-t+1])$ can be computed via Fact 3.1: start by computing $\mathsf{sk}_k(T[p-\ell'_j+1..n-t+1])$ from $\mathsf{sk}_k(Q)$, $\mathsf{sk}_k(Q[1..r])$, and the mismatch information for p and p', and then use $\mathsf{sk}_k(T[1..p-\ell'_j])$ to compute $\mathsf{sk}_k(T[1..n-t])$.
4. **Compute $\mathsf{hd}_{\le k}(T[t..n], T[1..n-t+1])$** using the computed sketches via Fact 3.1. If it is at most k, output t as a k-mismatch period, and return $w(T[t..])$ and $\mathsf{MI}(T[t..], T[..n-t+1])$.

▶ **Proposition 3.20.** *Assume that $j \ge 3$ and that P_j has more than $K = 576k$ k-mismatch occurrences in T_j. The algorithm computes $\mathcal{P}^j_k$, $\mathsf{MI}(T[p..], T[..n-p+1])$ and $w(T[p..])$ for $p \in \mathcal{P}^j_k$ in $O(n \cdot kt_w(n) \log^4 n)$ time and $O(k^2 \log^2 n + s_w(n))$ space and is correct w.h.p.*

Proof. The preprocessing of P_j takes $O(n \cdot (k \log^4 n + t_w(n)))$ time and $O(k \log^2 n + s_w(n))$ space. The main phase of the algorithm starts with an extension procedure (Algorithm 1), which takes $O(n(k \log^2 n + t_w(n)))$ total time and $O(k \log n + s_w(n))$ space. If the return is

executed in Line 6 of Algorithm 1, the algorithm then runs two instances of the k-mismatch algorithm (Corollary 3.3) that take $O(nk \log^4 n)$ time and $O(k \log^2 n)$ space. Adding elements to Pref_j and Suf_j, as well as look-ups, takes $O(1)$ time per element, and we add at most $K = O(k)$ elements in total. The two hash tables occupy $O(k^2 \log^2 n)$ space (Fact 3.8, Fact 3.1). Finally, testing all candidate positions requires $O(K \cdot k \log^3 n)$ time and $O(k \log n)$ space (Proposition 3.6). If the return is executed in Line 7 of Algorithm 1, the algorithm runs an instance of the k-mismatch algorithm, which takes $O(nk \log^4 n)$ time and $O(k \log^2 n)$ space, and maintains a constant number of sketches, taking $O(n \log^2 n)$ time and $O(k \log n)$ space. The process to test the candidate k-periods uses $O(k \log n + s_w(n))$ space and $O(k(\log^3 n + t_w(n)))$ time, and it is iterated $\leq n$ times.

The algorithm can fail if the preprocessing fails, if the k-mismatch algorithm errs, or if adding an element to the hash tables takes more than constant time, or if the test fails. By the union bound, Corollary 3.3, Fact 3.1, and Proposition 3.6, the failure probability is inverse-polynomial in n. ◀

Theorem 2.10 follows from Proposition 3.10, Proposition 3.20, and Proposition 3.14.

References

1 Gabriel Bathie, Tomasz Kociumaka, and Tatiana Starikovskaya. Small-space algorithms for the online language distance problem for palindromes and squares. In *ISAAC 2023*, volume 283 of *LIPIcs*, pages 10:1–10:17, 2023. doi:10.4230/LIPICS.ISAAC.2023.10.

2 Michael A. Bender, Martín Farach-Colton, John Kuszmaul, and William Kuszmaul. Modern hashing made simple. In *SOSA 2024*, pages 363–373, 2024. doi:10.1137/1.9781611977936.33.

3 Petra Berenbrink, Funda Ergün, Frederik Mallmann-Trenn, and Erfan Sadeqi Azer. Palindrome recognition in the streaming model. In *STACS 2014*, volume 25, pages 149–161, 2014. doi:10.4230/LIPIcs.STACS.2014.149.

4 Sudatta Bhattacharya and Michal Koucký. Locally consistent decomposition of strings with applications to edit distance sketching. In *STOC 2023*, pages 219–232, 2023. doi:10.1145/3564246.3585239.

5 Sudatta Bhattacharya and Michal Koucký. Streaming k-edit approximate pattern matching via string decomposition. In *ICALP 2023*, volume 261 of *LIPIcs*, pages 22:1–22:14, 2023. doi:10.4230/LIPICS.ICALP.2023.22.

6 Sudatta Bhattacharya and Michal Koucký. Locally consistent decomposition of strings with applications to edit distance sketching, 2023. doi:10.48550/arXiv.2302.04475.

7 Robert S. Boyer and J Strother Moore. MJRTY: A fast majority vote algorithm. In *Automated Reasoning: Essays in Honor of Woody Bledsoe*, Automated Reasoning Series, pages 105–118. Kluwer Academic Publishers, 1991.

8 Panagiotis Charalampopoulos, Tomasz Kociumaka, and Philip Wellnitz. Faster approximate pattern matching: A unified approach. In *FOCS 2020*, pages 978–989. IEEE, 2020. doi:10.1109/FOCS46700.2020.00095.

9 Raphaël Clifford, Tomasz Kociumaka, and Ely Porat. The streaming k-mismatch problem. In *SODA 2019*, pages 1106–1125. SIAM, 2019. doi:10.1137/1.9781611975482.68.

10 Richard Cole and Ramesh Hariharan. Approximate string matching: A simpler faster algorithm. In Howard J. Karloff, editor, *SODA 1998*, pages 463–472. ACM/SIAM, 1998. URL: http://dl.acm.org/citation.cfm?id=314613.314827.

11 Mohamed G. Elfeky, Walid G. Aref, and Ahmed K. Elmagarmid. Stagger: Periodicity mining of data streams using expanding sliding windows. In *ICDM 2006*, pages 188–199, 2006. doi:10.1109/ICDM.2006.153.

12 Funda Ergün, Elena Grigorescu, Erfan Sadeqi Azer, and Samson Zhou. Streaming periodicity with mismatches. In *APPROX/RANDOM 2017*, volume 81 of *LIPIcs*, pages 42:1–42:21, 2017. doi:10.4230/LIPICS.APPROX-RANDOM.2017.42.

13 Funda Ergün, Elena Grigorescu, Erfan Sadeqi Azer, and Samson Zhou. Periodicity in data streams with wildcards. *Theory Comput. Syst.*, 64(1):177–197, 2020. doi:10.1007/S00224-019-09950-Y.

14 Funda Ergun, Hossein Jowhari, and Mert Sağlam. Periodicity in streams. In *APPROX/RANDOM 2010*, pages 545–559, 2010.

15 Funda Ergün, Elena Grigorescu, Erfan Sadeqi Azer, and Samson Zhou. Streaming periodicity with mismatches, 2017. arXiv:1708.04381.

16 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.

17 Tomohiro I. Longest Common Extensions with Recompression. In *CPM 2017*, volume 78 of *LIPIcs*, pages 18:1–18:15, 2017. doi:10.4230/LIPIcs.CPM.2017.18.

18 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.

19 Gad M. Landau and Uzi Vishkin. Fast string matching with k differences. *Journal of Computer and System Sciences*, 37(1):63–78, 1988. doi:10.1016/0022-0000(88)90045-1.

20 Markus Lohrey. Algorithmics on slp-compressed strings: A survey. *Groups Complex. Cryptol.*, 4(2):241–299, 2012. doi:10.1515/GCC-2012-0016.

21 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.

22 Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *J. Algorithms*, 51(2):122–144, 2004. doi:10.1016/J.JALGOR.2003.12.002.

A A gap in the previous streaming algorithm for computing k-mismatch periods

In this section, we point to the specific claim in the correctness proof of the previous streaming algorithm for computing k-mismatch periods [15] which is, in our opinion, not true.

Let S be a string length n, and $1 \leq p < q \leq \frac{n}{2}$ two k-mismatch periods of S such that $k \cdot (p+q) \leq i \leq \frac{n}{2} - k \cdot (p+q)$, where i is a position of S, and $q \geq (2k+1) \cdot \mathsf{gcd}(p,q)$. The construction of [15] introduces a grid defined over a set $\{-k, \ldots, k\}^2$. A node (a,b) of the grid represents a position $i + ap + bq$ of S. For a node representing a position j, we add edges connecting it to nodes representing $j+p$, $j+q$, $j-p$, and $j-q$ (if they exist in the grid). Finally, we say that an edge (i,j) of the gird is *bad* if $S[i] \neq S[j]$.

The proof of [15, Theorem 28], one of the key elements of the streaming algorithm of Ergün et al., relies on the fact that there are a few bad edges in the grid. One of the steps in their proof of this fact is the following claim:

▶ **Proposition A.1** ([15, Claim 20]). *The nodes of the grid correspond to distinct positions of S.*

As an immediate corollary, and since p and q are both k-mismatch periods of S, they immediately derive that there are at most $2k$ bad edges in the grid.

The authors then extend their approach to the case when S has $m \in \mathbb{N}$ k-mismatch periods $p_1 < \cdots < p_m$, where $p_m \geq (2k+1) \cdot \mathsf{gcd}(p_1, p_m)$. One can construct an m-dimensional grid in a similar way to the case $m = 2$, and it is claimed that in this grid, "the total number of bad edges is at most mk" (Page 21). However, consider the string $S = \mathtt{a}^{40}\mathtt{ba}^{60}$. All integers smaller than 50 are 2-mismatch periods of S, and in this example we have $m = 50$, $k = 2$ and $\mathsf{gcd}(1, 50) = 1$, verifying the assumption $50 \geq (2k+1) \cdot \mathsf{gcd}(1, 50)$. Let $\{-2, \ldots, 2\}^{50}$ be the grid centred around the index 41 (i.e the only character $\mathtt{b}$). Similarly to the case $m = 2$, a point $(a_1, \ldots, a_{50})$ of the grid represents the position $41 + \sum i \cdot a_i$, and an edge

between nodes representing positions i, j is bad if $S[i] \neq S[j]$. Note for any $i \leq 25$, the node $(a_1, \ldots, a_{50})$ with $a_i = 2$, $a_{2i} = -1$ and $a_j = 0$ for $j \notin \{i, 2i\}$ represents the position $41 + 2i - 2i = 41$. As a result, 41 is represented by at least distinct 25 different nodes in the grid, which contradicts Proposition A.1 for $m \neq 2$. Furthermore, these nodes are connected with each of their neighbours with a bad edge. Since each node has at least 5 neighbours, there are at least 125 distinct bad edges, contradicting the upper bound on the number of bad edges which was $mk = 100$.

Parallel Joinable B-Trees in the Fork-Join I/O Model

Michael T. Goodrich
University of California, Irvine, CA, USA

Yan Gu
University of California, Riverside, CA, USA

Ryuto Kitagawa
University of California, Irvine, CA, USA

Yihan Sun
University of California, Riverside, CA, USA

Abstract

Balanced search trees are widely used in computer science to efficiently maintain dynamic ordered data. To support efficient set operations (e.g., union, intersection, difference) using trees, the *join-based* framework is widely studied. This framework has received particular attention in the parallel setting, and has been shown to be effective in enabling simple and theoretically efficient set operations on trees. Despite the widespread adoption of parallel join-based trees, a major drawback of previous work on such data structures is the inefficiency of their input/output (I/O) access patterns. Some recent work (e.g., C-trees and PaC-trees) focused on more I/O-friendly implementations of these algorithms. Surprisingly, however, there have been no results on bounding the I/O-costs for these algorithms. It remains open whether these algorithms can provide tight, provable guarantees in I/O-costs on trees.

This paper studies efficient parallel algorithms for set operations based on search tree algorithms using a join-based framework, with a special focus on achieving I/O efficiency in these algorithms. To better capture the I/O-efficiency in these algorithms in parallel, we introduce a new computational model, the Fork-Join I/O Model, to measure the I/O costs in fork-join parallelism. This model measures the total block transfers (I/O work) and their critical path (I/O span). Under this model, we propose our new solution based on B-trees. Our parallel algorithm computes the union, intersection, and difference of two B-trees with $O(m \log_B(n/m))$ I/O work and $O(\log_B m \cdot \log_2 \log_B n + \log_B n)$ I/O span, where n and $m \leq n$ are the sizes of the two trees, and B is the block size.

2012 ACM Subject Classification Computing methodologies → Massively parallel algorithms

Keywords and phrases Parallel algorithm, I/O efficiency, search trees, B-trees

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.37

Related Version *Full Version*: `https://arxiv.org/abs/2510.20053`

1 Introduction

Balanced search trees are among the most fundamental data structures in computer science. The search tree structure effectively maintains the ordering for a set of elements with dynamic updates, and a balance guarantee, usually meaning to bound the tree height to be logarithmic, enables efficient cost bounds for both updates and queries. Balanced search trees have been used to support basic data types such as ordered sets and maps in various programming languages, either as built-in data types or in standard libraries.

In addition to individual updates such as insertions and deletions, set operations (e.g., set-union, corresponding to merging two trees) are often needed in the interface of ordered sets and maps. To support efficient set operations on trees, many existing papers study the *join-based framework*. The core of the framework is a function *join*. $\mathsf{Join}(T_L, k, T_R)$ takes

© Michael T. Goodrich, Yan Gu, Ryuto Kitagawa, and Yihan Sun;
licensed under Creative Commons License CC-BY 4.0

two search trees T_L and T_R, and one key k in the middle, such that $\forall k_1 \in T_L$ and $k_2 \in T_R$, $k_1 < k < k_2$, and returns a valid balanced tree containing all elements in $T_L \cup \{k\} \cup T_R$. Existing work has shown that abstracting the Join primitive allows for elegant and efficient designs for set operations on various types of balanced trees, both sequentially [1, 2] and in parallel [7, 9, 10, 13, 14, 24].

The join-based framework was first studied in the sequential setting, on red-black trees [25] and weight-balanced trees [1]. The idea then has received particular attention in the parallel setting, since it conceptually allows for applying a batch of updates to the tree. Blelloch and Reid-Miller [10] first used the join-based framework in parallel on treaps, and proved that they are theoretically efficient. This idea was extended to various balancing schemes in 2014 on a data structure called P-tree [7, 23] and later implemented in a parallel library [24]. The theoretical results and implementations of these trees have been used both to support strong theoretical bounds in other parallel algorithms [12, 15, 17, 21], as well as in various applications such as databases [23], graph processing [14, 16], and more [22, 26].

Despite the widespread adoption of parallel join-based trees, a major drawback of previous work on such data structures is the inefficiency of their input/output (I/O) access patterns; e.g., see [4]. For example, in a tree node that only contains a few elements in the set, the number of memory accesses is asymptotically the same as the operations applied, rather than being a function of a block size or cache-line size. In fact, many follow-up works (e.g., C-tree [14] and CPAM [13]), aim to improve the memory-access efficiency (i.e., the I/O efficiency) with blocked tree nodes or leaves. While these new data structures have provable guarantees in work and span, and have shown good performance in experiments due to better memory access patterns, we are unaware of any theoretical improvements in their I/O efficiency. It is therefore worth asking, whether we can adapt the simple join-based framework to also achieve I/O-efficiency with provable guarantees.

This paper studies efficient parallel algorithms for set operations based on search tree algorithms using a join-based framework, with a special focus on achieving I/O efficiency in these algorithms. Our new solution is based on B-trees, and includes novel ideas to overcome challenges in both algorithm design and analysis for this problem. The challenge to achieving I/O-efficiency in parallel is two-fold. The first reason, which lies in the algorithm design aspect, is to apply the idea to an I/O-friendly data structure, such as B-trees, when binary structures are the norm. As discussed above, there exist previous studies that attempt to make join-based algorithms I/O-friendly by grouping multiple keys as a block in the same tree node or in the leaves [13, 14]. However, they tend to still keep the binary structure of the tree, which introduces needless complications. This design preserves compatibility with the existing join-based algorithmic framework, which can be used out of the box with minimal adaptation. However, since the tree height is $O(\log(n/B))$ for block size B and tree size n, this design does not lead to an ideal I/O bound for these set algorithms. To achieve non-trivial bounds, allowing multi-way trees (e.g., B-trees) is critical. However, the multi-way structure of these structures inevitably introduces complications into algorithm design. In particular, a multi-way join is required, which basically concatenates a set of trees with keys in between. Accordingly, a multi-split algorithm is needed to divide a tree using multiple splitter keys. Both of them need careful algorithm re-design over the existing ideas.

The second, and perhaps most interesting, reason related to the analysis aspect is the lack of an effective model to capture the I/O efficiency in these algorithms in parallel. While both models for parallel computing (e.g., PRAM) and I/O models have a long history, analyzing the I/O cost in parallel has some inherent difficulties such as shared/separate cache, synchronization schemes, exclusive/concurrent writes, etc. In 2010, Arge, Goodrich, and Sitchinava introduce the parallel external-memory (PEM) model [5], which focuses on

the I/O bottleneck in PRAM algorithms. Built on top of the PRAM model, PEM assumes synchronized threads. On the other hand, the join-based tree algorithms, as well as their implementations in software libraries, are based on the classical fork-join model, which is highly asynchronous.

Our Results

To accurately analyze I/O cost in asynchronous parallel algorithms, we introduce the *Fork-Join I/O Model* in this paper, which more formally defines the I/O cost in fork-join parallelism. Analogous to work and span for the standard fork-join model, the Fork-Join I/O Model measures the I/O cost of parallel algorithms by both the total number of block transfers (referred to as **I/O work**), and the maximum number of block transfers one depends on the previous (referred to as **I/O span**). Based on this model, we provide and analyze new parallel and I/O-efficient algorithms on set/map operations using B-trees.

Intuitively, the Fork-Join I/O Model provides a software-based asynchronous alternative to the hardware-based synchronous parallel external-memory (PEM) model. We also present parallel I/O-efficient algorithms for B-trees in this model, such as union, intersection, and difference operations. To do this, we first design a B-way join algorithm, which may concatenate at most B trees. Using this primitive, we design a general join algorithm that takes any number of k trees and $k-1$ keys in between, and an inverse function, that splits a tree by $k-1$ keys. Based on the two functions, we design an I/O-efficient parallel set algorithms. We also show in the Appendix how to modify the union algorithm to also perform the intersection and difference operations, while achieving the same I/O bounds asymptotically.

We highlight that the design of the algorithm is highly non-trivial. Directly applying the simple B-way join (and a corresponding B-way split) algorithm to a Union algorithm leads to $O(\log_B n \log_B m)$ I/O span. A specific interesting technical contribution in this paper is to achieve a stronger I/O span bound of $O(\log_B m \cdot \log_2 \log_B n + \log_B n)$, for which we introduce the more sophisticated algorithms of arbitrary-way join and split. We summarize the main results below.

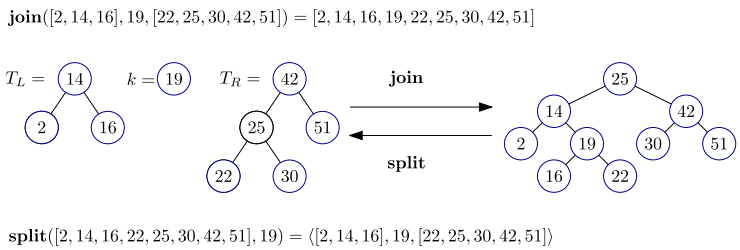
▶ **Theorem 1.1** (Parallel Set Operations on B-trees). *Given two B-trees with sizes m and $n \geq m$, there exists a parallel algorithm that returns a new B-tree containing the union of the two input trees in and $O\big(m \log_B\big(\frac{n}{m}\big)\big)$ I/O work, $O(\log_B m \cdot \log_2 \log_B n + \log_B n)$ I/O span, where B is the block size.*

Due to space limit, we present the Union algorithm and analyze it in the main paper, and show how to achieve the same bounds for Intersection and Difference in Appendix I.

2 Preliminaries

B-trees. A B-tree is a multi-way self-balancing tree data structure that maintains an ordered set of keys. With clear context, for a B-tree T, we also use T to denote the set of keys in T. We use the standard definitions of parent, child, sibling, ancestor, and descendants in B-trees. More formally, a B-tree is either an empty node (external node), or a node with a set of b keys $k_1, k_2, \ldots k_b$ with $b+1$ subtrees $T_1, T_2, \ldots T_{b+1}$ with the following invariants:

- $\lceil B/2 \rceil \leq b \leq B$ for some parameter B, except for the root, where $2 \leq b \leq B$.
- Each subtree T_i is a B-tree.
- $\forall x \in T_{i-1}$ and $y \in T_i$, we have $x < k_i < y$.
- All external nodes have the same depth, where the depth of a node is the number of ancestors (inclusive) of it.




```
union(T1, T2) {
    T_{L,1} ← left subtree of T1
    T_{R,1} ← right subtree of T1
    r ← the root of T1
    ⟨T_{L,2}, k, T_{R,2}⟩ ← split(T2, r)
    In parallel:
        L ← union(T_{L,1}, T_{L,2})
        R ← union(T_{R,1}, T_{R,2})
    return join(L, r, R)
}
```

Figure 1 Illustration for Join and Split algorithms on binary trees, and an example of the set union algorithm. To extend the idea to B-trees, multi-way Join and Split algorithms are needed.

The height of a B-tree T, denoted as $h(T)$, is 0 if it is an external node, and $h(T') + 1$ if T has a child T'. If T has multiple children, all of them should have the same height.

When we discuss an algorithm on two B-trees, we always use n to denote the one with the larger size and $m \leq n$ the smaller size.

The B-tree includes an operation for combining nodes when the number of keys in the node is below $\lceil B/2 \rceil$, and for splitting a node into two when there are more than B keys in the node. We will refer to these operations as **fuse** and **divide** operations, to avoid conflicting terminology.

Sequence Notation. We use $a_{i..j}$ $(i \leq j)$ to denote a sequence of elements $\langle a_i, a_{i+1}, \ldots, a_j \rangle$.

Background of Join-based Algorithms. For completeness, we introduce some background of join-based algorithms on binary trees in Appendix A. We present Figure 1 as an illustration for the high-level idea, as well as an example of implementing a Union based on join. We introduce more details about related work in Appendix B.

One technical challenge to extend this idea to B-trees is to support multi-way Join and Split functions, which may join multiple trees with keys in the middle, or split a tree using multiple splitters. Since the input tree heights may vary, these primitives must rebalance the returned tree properly. We introduce our algorithms for these primitives in Section 4.2 and 4.3, and finally present our Union algorithm in Algorithm 4.

3 The Fork-Join I/O Model

In this paper, we propose the Fork-Join I/O Model, which is an extension of the classic work-span model to analyze algorithms. The goal of this paper to use this model is to more precisely capture the I/O costs in algorithms based on fork-join parallelism, which is crucial for analyzing and implementing existing join-based algorithms on trees.

Recall that in the work-span model in the classic multithreaded model [6,9,11], we assume a set of threads share their memory. Each thread acts like a sequential RAM plus a fork instruction that forks two threads running in parallel. When both threads are finished with thir task, the original thread continues. A parallel-for is simulated by forks in a logarithmic number of steps. A computation can be viewed as a DAG. The *work* W of a parallel algorithm is the total number of operations, and the *span (depth)* S is the longest path in the DAG. In practice, we can execute the computation with work W and span S using a randomized work-stealing scheduler [11,18] in time $W/P + O(S)$ with P processors with high probability.

The extension we make to this model to adapt for practicality is that, instead of considering executing one word on RAM, we consider unit cost for reading or writing a block of size B, where B is a given parameter. Here B can be the cacheline size when referring to in-memory

algorithms, or the block size when considering external-memory algorithms. In either case, this is the basic unit for block transfers, which is indeed the motivation of the design of B-trees that improves the performance as compared to a simpler binary search tree. Here we assume each processor can hold $O(\log n)$ blocks (i.e., the stack memory) to execute the program. Accessing the data in these blocks is free (it does not asymptotically change the complexity, but only makes the analysis easier). We define this modified work as the **I/O work**. Regarding the span, we also adapt similarly (i.e., count the access of an entire block with unit cost), and refer to this as the **I/O span**. Here we assume a fork or join can create or synchronize B threads for simplicity. All I/O span bounds remain the same when translating the bound to PRAM, or are affected by a factor of $\log B$ (mostly a small constant) to the binary fork-join model [9].

An important subroutine we will be using in this model is the Gather primitive. This operation allows us to gather the $O(B)$ elements which are stored in various blocks in external memory into a single block in $O(1)$ I/O span and $O(B)$ I/O work. During the synchronization of each thread, the thread may return a $O(1)$ amount of data, thus the total amount of data returned by all threads is $O(B)$.

4 Parallel Joinable B-Tree

In this section, we present primitive algorithms for the joinable B-trees. We first introduce the B-Way-Join operation, which takes a sequence of $(b+1)$ B-trees and b keys in between, and combines them into a valid, balanced B-tree. We then show the B-Way-Join algorithm with detailed cost analysis in Section 4.1. Then, the B-Way-Join algorithm is used as a subroutine to develop the more general Multi-Split and Multi-Join algorithms in Section 4.2 and 4.3 in order to achieve better I/O bounds for the Union algorithm, described in Section 5.

These join and split algorithms are similar to the approach found in the other parallel search trees [7, 10], but adapted for the Fork-Join I/O Model.

These algorithms will be used as the building blocks for the set operations in Section 5. The key idea is to use the Multi-Split operation to split the two B-trees into $\sqrt{n+m}$ subtrees each, where the union of a pair of the subtrees is of size $\sqrt{n+m}$. We then make recursive calls to the union of these pairs, leading to $\sqrt{n+m}$ total subtrees, which we then use the Multi-Join algorithm to combine into a single B-tree.

The Multi-Split operation uses B-Way-Join as a primitive, by first searching for one of the keys they will be splitting on, which creates a series of subtrees which must then be joined together, rebalancing the resulting tree. The Multi-Join operation on the other hand is a more complex operation and differs significantly from the B-Way-Join operation.

4.1 B-way Join

4.1.1 Algorithm Description

The join operation merges $b+1$ B-trees $T_1, T_2, \ldots, T_{b+1}$, and b separator keys $k_1, k_2, \ldots, k_b$, into a single valid B-tree, where $b \leq B$. Each input tree T_i must be a valid B-tree and T_i contains keys only in the range (k_{i-1}, k_i), where $k_0 = -\infty$ and $k_{b+1} = \infty$.

The primary challenge lies in ensuring the join operation is both computationally efficient and maintains the balance of the tree, even when the height of the input trees may differ significantly. We describe our algorithm below in three steps. An illustration of the three steps on an example input is in Section 4.1. In Appendix C, we prove the correctness and efficiency of this algorithm.

Algorithm 1 B-Way-Join$(T_{1..(b+1)}, k_{1..b})$.

Input: A sequence of $b+1$ B-trees $T_{1..(b+1)}$ and b keys $k_{1..b}$. For any $x \in T_i$ and $y \in T_{i+1}$, we have $x < k_i < y$.
Output: A B-tree T with all keys in $\{k_1, k_2, \ldots, k_b\} \cup \bigcup T_i$

```
1  h* ← max_i h(T_i)                                   // h* is the largest tree height
2  A ← the subsequence of ⟨1, ..., b⟩, where i ∈ A iff. h(T_i) = h*   // indexes of tall trees
3  if the first element in A is not 1 then add 0 at the beginning of A
4  ParallelForEach i ∈ A do
5      i' ← the successor of i in A                    // T_{i'} is the next tree with height h*
6      if i = 0 then // Handling the special case when there exist keys to the left of the first tall tree
7          R ← the leftmost subtree of T_{i'}
8          S ← ⟨T_1, ... T_{i'-1}, R⟩                  // The sequence of trees for the leftmost recursive call
9          k' ← ⟨k_1, k_1, ..., k_{i'-1}⟩              // The sequence of keys for the leftmost recursive call
10         X ← B-Way-Join(S, k')                       // All keys to the left of the first tall tree
11         Replace the leftmost subtree of T_{i'} with X   // T_{i'} may be unbalanced for now
12     else
13         L ← the rightmost subtree of T_i
14         S ← ⟨L, T_{i+1}, T_{i+2}, ..., T_{i'-1}⟩    // The sequence of trees for the recursive call
15         k' ← ⟨k_{i+1}, k_{i+2}, ..., k_{i'-1}⟩      // The sequence of keys for the recursive call
16         X ← B-Way-Join(S, k')   // X will contain all keys between the i-th and the i'-th tree
17         Replace the rightmost subtree of T_i with X     // T_i may be unbalanced for now
18 if the first element in A is 0 then remove 0 from A
19 T ← Concatenate all (new) trees T_i, with keys k_{i-1}, for all i ∈ A
20 SimpleRebalance(T)                                  // Introduced in Lemma 5
```

Step 1: Grouping. The algorithm starts by identifying the height of the tallest tree in the input set, which we denote as h^*. We say a tree T_i is a **tall tree** if it has height h^*, and a **short tree** otherwise. Note that there may be multiple tall trees in the sequence T. Let $A \subseteq \langle 0, 1, \ldots, b\rangle$ be the subsequence of indices of trees with height h^*. We then partition the input trees into $|A|+1$ groups. At a high level, each group contains all keys and trees between two tall trees, and the leftmost subtree of the previous tall tree. If T_1 is not a tall tree, a special group, which we call a **leading group**, to the left of T_1 will also be formed.

More formally, assume $i \in A$, and $i' = \mathsf{succ}(i)$ is the successor of i in A.[1] Then the corresponding group includes $i' - i$ trees, which are the rightmost subtree in T_i, and all trees from T_{i+1} to $T_{i'-1}$, and $i' - i - 1$ keys, which are from k_i to $k_{i'-2}$. Note that the trees are separated by the corresponding keys in the same group. If T_1 is not a tall tree, all elements to the left of the first tall tree $T_{i'}$ form the leading group, which includes all trees up to $T_{i'-1}$, the leftmost subtree of $T_{i'}$, and all keys up to $k_{i'-1}$.

Step 2: Recursing and Subtree Replacing. After the first step, each group contains a sequence of trees, and the corresponding keys that separate them. The second step simply calls B-Way-Join recursively to reorganize each group into a valid B-tree. After that, for the group corresponding to trees between T_i and $T_{\mathsf{succ}(i)}$, we use the resulting tree to replace the rightmost subtree in T_i. If there exists a leading group, it replaces the leftmost subtree in the first tall tree.

In Theorem 7 we will show that B-Way-Join on a sequence of trees with maximum height h will lead to a tree with height at most $h+1$. In this case, all trees involved in the recursive call is either a short tree, which has height at most $h^* - 1$, or a subtree of a tall tree, which

[1] We define $\mathsf{succ}(\max(A))$, where $\max(A)$ the last element in A, to be $b+1$.

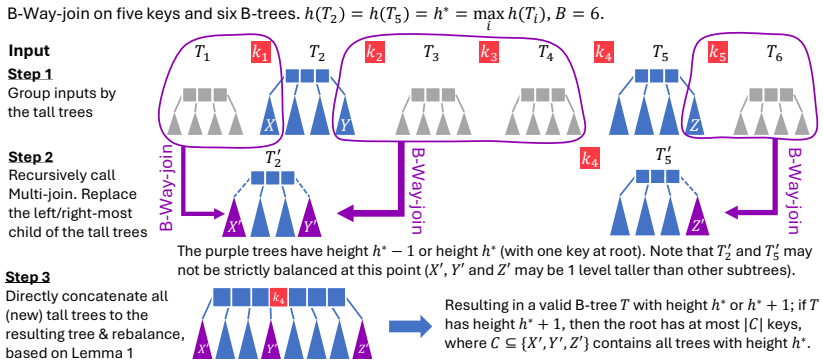


Figure 2 An illustration of the B-Way-Join algorithm (described in Section 4.1) with 6 trees and 5 keys. The blue trees are the *tall trees* with height h^*. All other trees has height less than h^*.

has height $h^* - 1$. Therefore, the resulting trees from recursive calls have height at most h^*. After replacing a subtree in a tall tree, it can be at most one level taller than the other subtrees.

Step 3: Concatenating and Rebalancing. After the previous step, all short trees have been combined into a tall tree. Note that for each tall tree, except for the first one, the key immediately before it is not included in any groups. Therefore, our final step is to concatenate all these remaining components together, which includes all tall trees T_i for all $i \in A$, and k_{i-1} for all $i \in A$. This is performed by creating a B-tree node with keys at the root at T_{A_1}, then k_{A_2-1}, then all keys at the root of T_{A_2}, then k_{A_3-1}, so on so forth. The subtrees, from left to right, are all subtrees in T_{A_1}, then all subtrees in T_{A_2}, so on so forth. Let the resulting tree be T.

The tree T is a valid B-tree except for two aspects: 1) some of its subtrees may be taller than other subtrees, but can be taller by at most 1, and 2) the number of keys at the root may be more than B. For such minor imbalance, we will use the SimpleRebalance algorithm, introduced in Lemma 5, to rebalance the tree. At a high level, the rebalancing algorithm simply promotes all keys at the subtree root of the taller subtrees to the root of the entire tree, and if an overflow occurs, we can promote every B keys at the root to a higher level, increasing the tree height by 1.

See Figure 2 for an illustration of the algorithm.

▶ **Theorem 4.1** (B-Way Join Analysis). *Let $T_1, T_2, \ldots, T_{b+1}$ be a set of B-trees, with the largest tree height $h_{\max}$ and the shortest tree height $h_{\min}$, and $k_1, k_2, \ldots, k_b$ be a set of separator keys, where $b \leq B$. The B-Way-Join operation can be performed in $O(h_{\max} - h_{\min})$ I/O span and $O(b \cdot (h_{\max} - h_{\min}))$ I/O work.*

We analyze the cost of the B-Way-Join operation in Appendix C.

4.2 Multi-Split

The Multi-Split algorithm takes a B-tree T and a sorted list of d keys $k_{1..d}$, and returns $d+1$ subtrees, such that the i-th subtree contains all keys in the range (k_{i-1}, k_i), where $k_0 = -\infty$ and $k_{d+1} = \infty$. This is achieved by spawning $d+1$ threads, where each thread i invokes a helper function Thread-Split, which outputs a single subtree containing all keys in the range (k_{i-1}, k_i). See Algorithm 2.

Algorithm 2 Thread-Split(T, k_1, k_2).

Input: A B-trees T and two keys k_1, k_2 such that $k_1 < k_2$
Output: A B-tree T containing all keys in the range (k_1, k_2)

```
1  if T is empty or contains only keys between k1 and k2 then return T
2  if T is a leaf then return the B-tree containing all keys in the range (k1, k2)
3  r_{1..b} ← the keys at the root of T
4  i ← the smallest index such that k1 < r_i, 0 if k1 < r1
5  j ← the largest index such that r_j < k2, b + 1 if r_b < k2
6  C_{i..j} ← all children of T from i to j, inclusive
7  if i = j then return Thread-Split(C_i, k1, k2)
8  C'_i ← Thread-Split(C_i, k1, ∞)
9  C'_j ← Thread-Split(C_j, −∞, k2)
10 r'_{i..j} ← copy of keys r_{i..j}        // Prevents adjacent threads from dividing the same node
11 return B-Way-Join(⟨C'_i, C_{i+1..j−1}, C'_j⟩, r'_{i..j})
```

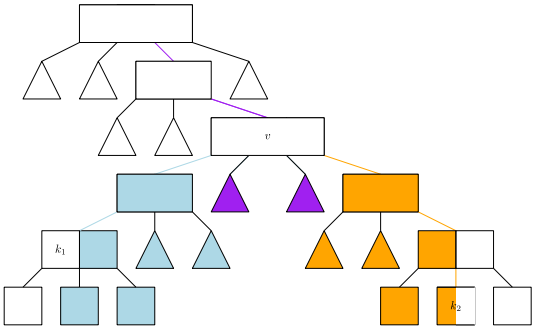


Figure 3 Above is a B-Tree, where each node rectangle is a node in the B-Tree, whereas the triangles are subtrees shortened for space. There are two search paths to k_1 and k_2 from the root, sharing the purple path at the start, then separating into the blue and orange paths respectively at node v. All nodes colored blue are the result of following the blue path and copying the appropriate keys, and similarly for the orange nodes. The node containing k_2 excludes all keys greater than or equal to k_2, thus being half orange. The result of Thread-Split will be the union of the blue, orange, and purple nodes. All white sections of the tree will be the search paths of other Thread-Split operations.

The high-level idea of the algorithm is to search for the keys k_i and k_{i+1} in T, copying the nodes and keys along the search path that fit within the range (k_i, k_{i+1}). Any node that lies on the shared prefix of both search paths, i.e. before the paths to k_i and k_{i+1} diverge, are not part of the output tree and can be skipped. Once the paths diverge, recursively split down the search path of k_1 and k_2. If our entire node falls within the range (k_1, k_2), then a copy of the pointer to the node is made. Otherwise, only the keys which fall within the range (k_1, k_2) and the pointers to the children that fall within the range are copied.

We copy the keys of the node instead of dividing it, which is the natural operation to perform, because dividing requires synchronization between adjacent threads. See Figure 3.

On the way back up the tree, the algorithm performs a B-Way-Join operation to combine the subtrees and keys which contain the keys in the range (k_1, k_2). This serves the purpose of both combining and rebalancing the subtrees. We prove the following theorem in Appendix D.

▶ **Theorem 1** (Multiway Split Cost). *We can split a B-tree T by $k_1, \ldots, k_d$ keys in parallel with $O(\log_B d + \log_B n)$ I/O span and $O(d \log_B n)$ I/O work, where n is $\sum_{i=1}^{d} |T_i|$, in the Fork-Join I/O Model.*

Algorithm 3 B-Way-Join-Fast$(T_{1..b+1}, k_{1..b})$.

Input: A sequence of $b+1$ B-trees $T_{1..b+1}$ and b keys $k_{1..b}$. For any $x \in T_i$ and $y \in T_{i+1}$, we have $x < k_i < y$. For all T_i, there is at most one node with B or more keys along their left and right spines.

Output: A B-tree T with all keys in $\{k_1, k_2, \ldots, k_b\} \cup \bigcup T_i$

$h^* \leftarrow \max_i h(T_i)$ *// h^* is the largest tree height*

$A \leftarrow$ the subsequence of $\langle 1, \ldots, b \rangle$, where $i \in A$ iff. $h(T_i) = h^*$ *// indexes of tall trees*

if the first key in A is not 1 **then**

  $i' \leftarrow$ the first key in A

  $T_{i'} \leftarrow$ Join-With-Tall$(T_{1..i'}, k_{1..i'-1})$

ParallelForEach $i \in A$ **do**

  $i' \leftarrow$ the successor of i in A *// $T_{i'}$ is the next tree with height h^**

  $T_{i'} \leftarrow$ Join-With-Tall$(T_{i..i'}, k_{i..i'-1})$

if the first key in A is 0 **then** remove 0 from A;

$T \leftarrow$ Concatenate all (new) trees T_i, with keys k_{i-1}, for all $i \in A$

SimpleRebalance(T) *// See Lemma 5*

4.3 Multi-Join

Suppose we have $d+1$ B-trees $T_{1,\ldots,d+1}$, and a sorted list of d keys $k_{1,\ldots,d}$, such that for any $x \in T_i$ and $y \in T_{i+1}$, we have $x < k_i < y$. The Multi-Join algorithm takes as input $T_{1,\ldots,d+1}$ and $k_{1,\ldots,d}$, and returns a single tree T, such that T contains $\bigcup_{i=1}^{d} T_i$ and $\{k_1, k_2, \ldots, k_d\}$.

We first describe here an overview of the Multi-Join algorithm. Section 4.1 shows how to join B trees in $O(\log_B n)$ I/O span and $O(B \log_B n)$ I/O work. Therefore, a simple solution is to join B trees together in parallel across $\log_B d$ rounds of joining. However, this results in an algorithm with $O(\log_B d \log_B n)$ I/O span, which is not optimal for larger values of d.

We improve the upper bound on the I/O span by reducing the cost of joining B trees down to an $O(\log_B \log_B n)$ I/O span and $O(\log_B n)$ I/O work, which we call the B-Way-Join-Fast operation. Let us define nodes containing more than B keys as **overloaded**, and nodes with exactly B keys as **full**. In order to achieve these bounds, we use a similar strategy to Akhremtsev and Sanders [3], where the trees are preprocessed such that each tree T_i contains a list of pointers to all nodes along their left and right spines. This allows us to efficiently find the level at which each tree must traverse in order to perform the join operation. For each B-Way-Join-Fast operation, the output tree must also maintain this list of pointers. Then we show how to divide all nodes that will be overloaded during B-Way-Join-Fast in parallel.

4.3.1 Faster B-way Join

Let L_i and R_i be the list of pointers to the left and right spines of some tree T_i respectively. L_i and R_i are sorted in reverse order by the height of the node, i.e. the leaf node is at index 0 and the root is at index $h(T_i)$. Implementation details are discussed in Section 4.3.2.

Step 1: Grouping As in the B-Way-Join algorithm in Section 4.1, we first identify the tallest trees of $T_{1,\ldots,b}$, and group the trees and keys together.

Step 2: Fusing Within Groups Then, we use the list of pointers to the left and right spines in order to jump directly to the correct height of the tree for each group.

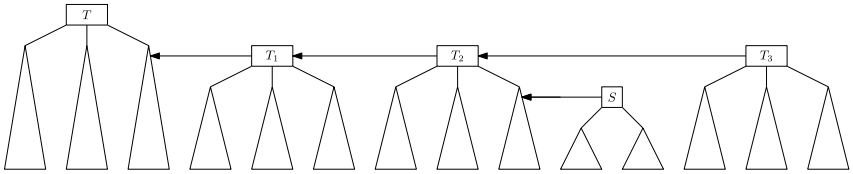


Figure 4 Trees T_1, T_2, and T_3 are all trees of the same height in the same group. T_1 fuses with a node in T, but at the same time, T_2 is also fusing with the root node of T_1, and T_3 is fusing with the root node of T_2. Therefore, there is a dependency where X must fuse with T_2 first, then T_2 must fuse with T_1, then T_1 with T. While S is also fusing with a node in T_2, S is fusing with node other than the root of T_2, thus there is no conflict.

Each tree T_j is assigned some T_i to be fused with, where i is the largest index within the same group such that $i < j$ and $h(T_i) \geq h(T_j)$. Then a thread is spawned for each T_j, and its corresponding key are fused with $L_i[h(T_j)]$ or $R_i[h(T_j)]$. As in Section 4.1, we also account for the special case where T_1 is not the tallest tree.

Conflicts occur when multiple trees of the same height are in the same group, since threads may try to fuse with roots of trees that are also fusing with other trees. However, in constant I/O span and work, each thread is able to identify which portion of the tree they will end up fusing with and directly write to that node. Consider the tree T_i, and let X be the set of all trees with the same height as T_i. X can be constructed using Gather to get the heights of all T_i in a single cache line and constructing each set in internal memory. Then all B threads are spawned, and thread i can check if $|X| > 1$. If so, thread i can also identify the tree in X with the smallest index and the tree T' they would be fusing with. If T' exists, then thread i can fuse T_i with T' instead, while ensuring there are no write conflicts with other threads which are fusing with the same node. For example, in Figure 4, T_1 is writing keys to some node in T, which we call v. T_2 will also write to v, but starting from the index $|v| + |T_1|$ away from the start of $|v|$.

Step 3: Dividing Large Nodes After each thread has finished fusing their trees, they check if the node is overloaded. If so, then the thread will divide the node into $\lceil \frac{b}{B} \rceil$ nodes, where b is the number of keys in the node. However, before pushing $\lceil \frac{b}{B} \rceil - 1$ keys up to the parent node, all threads are first synchronized in order to avoid write conflicts. Then all threads are spawned again to push up their keys to the parent node. We show in Lemma 10, in Appendix E, all overloaded nodes will push at most one key to their parent node.

Step 4: Dividing Smaller Nodes in Parallel Any remaining overloaded nodes will only cause consecutive full nodes above it to be divided, since a single key is pushed up per overloaded node. By computing the number of consecutive full nodes above each overloaded node, we can divide these nodes in parallel.

Using the prefix sum algorithm, described in Appendix F of the Appendix, we can accomplish this. Create an array C_v of length $\log_B n$ for each overloaded node v, where $C_v[i]$ is set to 1 if the node i levels above is full or overloaded, and 0 otherwise. If there is no node i levels above v, then $C_v[i]$ is set to 0. Then take the prefix sum of each C_v, and mark the node i levels above v to be divided if $C_v[i] = i$. The number of threads which marks a node to be divided is the number of keys pushed up into it. Finally, a thread is spawned for each node which is marked to be divided, and is divided in parallel.

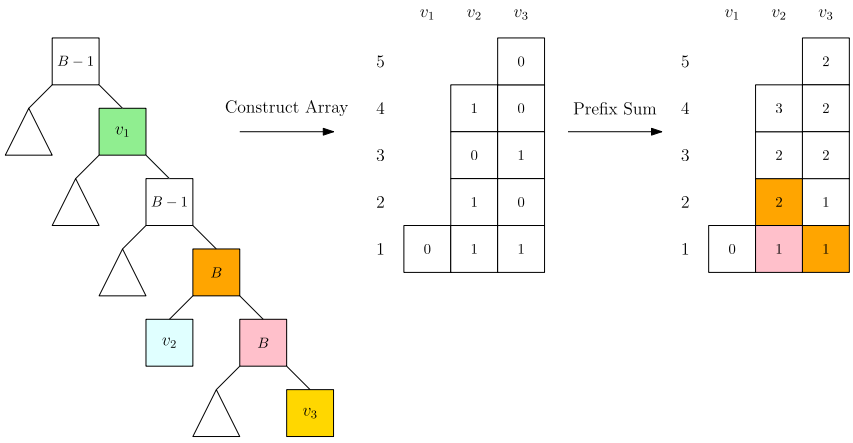


Figure 5 The B-tree contains 3 nodes v_1, v_2, and v_3 which are overloaded. We use the list of pointers in order to construct our arrays C. A cell within the array is set to 1 if the node is full or overloaded, i.e. it contains B or more keys. Otherwise, it is set to 0. Note that since v_1 only contains a single node above it, its array is also of length 1. Then we take the prefix sum of each array, and mark a node to be divided if $C[i] = i$. The three colored cells are the nodes which are to be divided. Two of the cells are orange, referring to the same node, meaning the node takes a key from two of its children.

Step 5: Concatenating and Rebalancing Just as in the B-Way-Join algorithm, after the previous steps, all short trees have been combined into a single tall tree, which is then rebalanced.

4.3.2 Maintaining the Left and Right Pointers

In order to achieve the I/O span and work bounds we desire, maintaining the list of pointers along the spines of the trees is crucial. We achieve this by implementing the lists as sequential B-trees, where the keys are pointers to the nodes along the spine. By using order statistics information, searching for the node at height i takes $O(\log_B \log_B n)$ I/O span and work.

If a new node is created for the left or right spine after the dividing process, we add the node to the front of L_i or R_i respectively. This also takes $O(\log_B \log_B n)$ I/O span and work. Finally, after the fusing operations complete, we must also update the pointers to the left/right spines of the trees. Recall in Figure 4 that part of the right spine of T_1 is being covered by the tree T_3, specifically all nodes with a height lower than T_3's root are not on the right spine of the output tree. More generally, suppose S is the set of all trees that were grouped together and has at least one node from their right spine which is part of the right spine of the output tree. Then $T \in S$ will have all nodes along its right spine less than the height of the next smaller tree in S covered. These are the nodes that should be removed from their list of spines. To do this, we simply find the height, h, of the next smaller tree in S and perform a sequential split on T. Then we use B-Way-Join to join all trees together.

5 Union

5.1 Algorithm Description

Let T_1 and T_2 be two B-trees with n and m keys respectively.

The union algorithm is a simple d-way divide and conquer algorithm, where $d = \sqrt{n+m}$. If $m < \sqrt{n+m}$, then we have reached the base case, and we call a I/O work inefficient union algorithm. Let $|T_1| \geq |T_2|$, as we can swap the two trees if this is not the case. The I/O work

Algorithm 4 Union(T_1, T_2).

Input: Two B-trees T_1 and T_2
Output: A B-tree T with all keys in $T_1 \cup T_2$

```
1  if |T2| > |T1| then T1, T2 ← T2, T1 ;
2  if |T2| < √(|T1| + |T2|) then return Work-Inefficient-Union(T1, T2) ;
3  d ← √(|T1| + |T2|)
4  ParallelForEach 1 ≤ i ≤ d do
5  |  k_i ← Two-Way-Binary-Search(((|T1|+|T2|)i)/d, T1, T2)
6  ⟨T_{1,1}, T_{1,2}, ..., T_{1,d+1}⟩ ← Multi-Split(T1, k_{1..d})
7  ⟨T_{2,1}, T_{2,2}, ..., T_{2,d+1}⟩ ← Multi-Split(T2, k_{1..d})
8  ParallelForEach 1 ≤ i ≤ d + 1 do
9  |  T_i ← Union(T_{1,i}, T_{2,i})
10 return Multi-Join(T_{1..d+1}, k_{1..d})
```

inefficient union algorithm uses all the keys in T_2 and uses them as keys to split on T_1 using the Multi-Split operation. Then we use the Multi-Join operation to combine the d subtrees and the $d-1$ keys we split on.

Otherwise, we search T_1 and T_2 for $d-1$ keys, which we will use to split each tree into d subtrees using the Multi-Split operation. We denote the subtrees of T_1 as $T_{1,\{1,\ldots,d\}}$ and T_2 as $T_{2,\{1,\ldots,d\}}$. The i-th key which we split on is the global $(n+m)\cdot i/d)$-th key. This ensures that $|T_{1,i}| + |T_{2,i}|$ is all approximately $\frac{n+m}{d}$. We then spawn $O(d)$ threads, and recursively call the union algorithm on each pair of subtrees $T_{1,i}$ and $T_{2,i}$. Then we use the Multi-Join operation to combine the d subtrees and the $d-1$ keys we split on.

5.2 Cost Analysis

Due to space constraints, we have put the details of the analysis for the base case of the union algorithm to Appendix G.

▶ **Lemma 2** (Union Algorithm Base Case Cost). *The Work-Inefficient-Union algorithm takes $O(\log_B m \cdot \log_2 \log_B n + \log_B n)$ I/O span and $O(m \log_B n)$ I/O work.*

▶ **Lemma 3.** *Algorithm 4 takes $O(\log_B m \cdot \log_2 \log_B n)$ I/O span.*

Proof. The I/O span of finding $d-1$, where $d = \sqrt{n+m}$, keys and splitting along them costs $O(\log_B(\sqrt{n+m}))$, and the I/O span of joining all the recursive components costs $O(\log_B(\sqrt{n+m}) \cdot \log_2 \log_B n)$. Recall that $m > \sqrt{n+m}$, since otherwise we would be in the base case of the algorithm. Therefore, the total I/O span of the above operations is $O(\log_B m \cdot \log_2 \log_B n)$.

Then for the next recursive call, the problem size shrinks to $\sqrt{n+m}$. The I/O span of the base case is also $O(\log_B m \cdot \log_2 \log_B n)$. Therefore, the cost of the I/O span is upper bounded by

$$S(n+m) = S(\sqrt{n+m}) + O(\log_B m \cdot \log_2 \log_B (n+m)).$$

Solving for this recurrence relation gives us $O(\log_B m \cdot \log_2 \log_B n)$, when $m < n$. ◀

▶ **Lemma 4.** *The total I/O work of the Union algorithm is $O(m \log_B \frac{n}{m})$.*

Proof. The total cost of the base case for the Union algorithm is $O(m \log_B \frac{n}{m})$ from Lemma 2 in the Appendix. Thus, we focus on the cost of the recursive calls.

We see that the I/O work cost is dominated by the cost of Multi-Split, Multi-Join, and searching for the split keys. All of these operations take $O(\sqrt{n+m}\log_B(n+m))$ I/O work.

We perform a similar analysis to the one in Blelloch et al. [9]. Consider the recurrence tree of this algorithm, which is a $\sqrt{n+m}$-way tree, and the cost of the current node is $\sqrt{n+m}\log_B(n+m)$. Since the recursion stops when $m < \sqrt{n+m}$, the recurrence tree is not balanced and some branches will reach the base case earlier than others. However, the cost of a node at the same level also costs the same, and decreases as the tree goes deeper. Let $t = n+m$, then the total I/O work of Union is:

$$t^{1/2}\log_B t + t^{1/2}\cdot t^{1/4}\log_B t^{1/2} + \cdots = \sum_i \frac{\log_B t}{2^i} t^{1-1/2^{i+1}}.$$

We can see the above recurrence is leaf dominated. For all i, $t^{1-1/2^{i+1}}$ is the number of split keys which are being searched. Naturally the total number of split keys can never be more than m, else we would have already reached the base case, meaning that $t^{1-1/2^{i+1}} \le m$. Let $x = t^{1/2^{i+1}}$, then the total leaf cost is

$$O(t^{1-1/2^{i+1}}\cdot\log_B t^{1/2^{i+1}}) = O\left(\frac{t}{x}\log_B x\right).$$

Clearly the above function decreases as x approaches infinity, and since $t^{1-1/2^{i+1}} \le m$, we know that $x \ge t/m$. Letting $x = t/m$ gives us the maximum value of the function, which is $O(m\log_B(\frac{n}{m}))$, giving us our desired I/O work bounds. ◀

Combining the results above we have the main theorem of this paper.

▶ **Theorem 1.1** (Parallel Set Operations on B-trees). *Given two B-trees with sizes m and $n \ge m$, there exists a parallel algorithm that returns a new B-tree containing the union of the two input trees in and $O\big(m\log_B\big(\frac{n}{m}\big)\big)$ I/O work, $O(\log_B m\cdot\log_2\log_B n + \log_B n)$ I/O span, where B is the block size.*

We show in Appendix I how to achieve the above bounds for the other set operations.

6 Conclusion

In this paper, we have introduced the parallel set operations on B-trees that achieve provable I/O efficiency in the context of a novel cost model, the Fork-Join I/O Model, which offers a way to analyze the I/O costs of algorithms in the fork-join setting. This captures the I/O complexity of algorithms executed under asynchronous fork-join parallelism, unlike the Parallel External Memory (PEM) model, which assumes synchronized threads at each step of the algorithm. Our model provides a more realistic abstraction for analyzing parallel algorithms that rely on asynchronous scheduling. We showed how to adapt the join-based paradigm to B-trees by developing new algorithms for Multi-Join, and Multi-Split. These primitives serve as building blocks for implementing more complex operations, such as the union, intersection, and difference operations. This allows us to achieve a near-optimal I/O work of $O\big(m\log_B\big(\frac{n}{m}\big)\big)$, and I/O span of $O(\log_B m\cdot\log_2\log_B n + \log_B n)$. Potential future work would be to reduce the I/O work to the optimal $O\big(\frac{m}{B}\log_B\big(\frac{n}{m}\big)\big)$ I/O complexity, and reduce the I/O span to $O(\log_B(n+m))$.

References

1 Stephen Adams. Implementing sets effciently in a functional language. Technical Report CSTR 92-10, University of Southampton, 1992.

2 Stephen Adams. Efficient sets – A balancing act. *Journal of functional programming*, 3(04):553–561, 1993. doi:10.1017/S0956796800000885.

3 Yaroslav Akhremtsev and Peter Sanders. Fast parallel operations on search trees. *CoRR*, abs/1510.05433, 2015. arXiv:1510.05433.

4 Yaroslav Akhremtsev and Peter Sanders. Fast parallel operations on search trees. In *IEEE International Conference on High Performance Computing (HiPC)*, 2016.

5 Lars Arge, Michael T. Goodrich, and Nodari Sitchinava. Parallel external memory graph algorithms. In *IPDPS*, pages 1–11. IEEE, 2010. doi:10.1109/IPDPS.2010.5470440.

6 Nimar S. Arora, Robert D. Blumofe, and C. Greg Plaxton. Thread scheduling for multiprogrammed multiprocessors. *Theory of Computing Systems*, 34(2):115–144, 2001. doi:10.1007/S00224-001-0004-Z.

7 Guy E. Blelloch, Daniel Ferizovic, and Yihan Sun. Just join for parallel ordered sets. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2016.

8 Guy E. Blelloch, Jeremy T. Fineman, Yan Gu, and Yihan Sun. Optimal parallel algorithms in the binary-forking model. *arXiv preprint 1903.04650*, 2019.

9 Guy E. Blelloch, Jeremy T. Fineman, Yan Gu, and Yihan Sun. Optimal parallel algorithms in the binary-forking model. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 89–102, 2020. doi:10.1145/3350755.3400227.

10 Guy E. Blelloch and Margaret Reid-Miller. Fast set operations using treaps. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 16–26, 1998. doi:10.1145/277651.277660.

11 Robert D. Blumofe and Charles E. Leiserson. Scheduling multithreaded computations by work stealing. *Journal of the ACM*, 46(5):720–748, 1999. doi:10.1145/324133.324234.

12 Nairen Cao and Jeremy T Fineman. Parallel exact shortest paths in almost linear work and square root depth. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 4354–4372. SIAM, 2023. doi:10.1137/1.9781611977554.CH166.

13 Laxman Dhulipala, Guy E. Blelloch, Yan Gu, and Yihan Sun. PaC-trees: Supporting parallel and compressed purely-functional collections. In *ACM Conference on Programming Language Design and Implementation (PLDI)*, 2022.

14 Laxman Dhulipala, Guy E Blelloch, and Julian Shun. Low-latency graph streaming using compressed purely-functional trees. In *ACM Conference on Programming Language Design and Implementation (PLDI)*, pages 918–934, 2019. doi:10.1145/3314221.3314598.

15 Laxman Dhulipala, David Eisenstat, Jakub Lacki, Vahab Mirrokni, and Jessica Shi. Hierarchical agglomerative graph clustering in poly-logarithmic depth. *arXiv preprint:2206.11654*, 2022.

16 Laxman Dhulipala, Charlie McGuffey, Hongbo Kang, Yan Gu, Guy E Blelloch, Phillip B Gibbons, and Julian Shun. Sage: Parallel semi-asymmetric graph algorithms for nvrams. *arXiv preprint*, 2019. arXiv:1910.12310.

17 Xiaojun Dong, Yan Gu, Yihan Sun, and Yunming Zhang. Efficient stepping algorithms and implementations for parallel shortest paths. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 184–197, 2021. doi:10.1145/3409964.3461782.

18 Yan Gu, Zachary Napier, and Yihan Sun. Analysis of work-stealing and parallel cache complexity. In *SIAM Symposium on Algorithmic Principles of Computer Systems (APOCS)*, pages 46–60. SIAM, 2022. doi:10.1137/1.9781611977059.4.

19 Simon Marlow et al. Haskell 2010 language report. *Available online*, 2010. URL: http://www.haskell.org/ (May 2011).

20 Kurt Mehlhorn and Stefan Näher. *LEDA: A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, New York, NY, USA, 1999.

21 Zheqi Shen, Zijin Wan, Yan Gu, and Yihan Sun. Many sequential iterative algorithms can be parallel and (nearly) work-efficient. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2022.

22 Yihan Sun and Guy Blelloch. Implementing parallel and concurrent tree structures. In *ACM Symposium on Principles and Practice of Parallel Programming (PPOPP)*, pages 447–450, 2019. doi:10.1145/3293883.3302576.

23 Yihan Sun, Guy E Blelloch, Wan Shen Lim, and Andrew Pavlo. On supporting efficient snapshot isolation for hybrid workloads with multi-versioned indexes. *Proceedings of the VLDB Endowment (PVLDB)*, 13(2):211–225, 2019. doi:10.14778/3364324.3364334.

24 Yihan Sun, Daniel Ferizovic, and Guy E Blelloch. PAM: Parallel augmented maps. In *ACM Symposium on Principles and Practice of Parallel Programming (PPOPP)*, 2018.

25 Robert Endre Tarjan. *Data Structures and Network Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1983.

26 Letong Wang, Xiangyun Ding, Yan Gu, and Yihan Sun. Fast and space-efficient parallel algorithms for influence maximization. *Proceedings of the VLDB Endowment (PVLDB)*, 17(3), 2023. doi:10.14778/3632093.3632104.

A Background for join-based Algorithms on Binary Trees

For completeness, we briefly introduce the background of join-based algorithms on binary trees. For binary trees, the $\mathsf{Join}(L, k, R)$ function takes two search trees T_L and T_R, and one key k in the middle, such that $\forall k_1 \in T_L$ and $k_2 \in T_R$, $k_1 < k < k_2$, and returns a valid balanced tree containing all elements in $L \cup \{k\} \cup R$. As the inversion to Join, a $\mathsf{Split}(T, k)$ is defined to return $\langle T_L, k', T_R \rangle$, where T_L (T_R) contains all keys in T smaller (larger) than k, and k' is the node containing k if $k \in T$, and null otherwise. In this case, an example of the Union algorithm can be implemented as illustrated in Figure 1. Note that the Union algorithm is independent of the balancing schemes used, as long as the Join and Split functions are properly supported.

B Related Work

The Join operation was first proposed by Tarjan [25] on red-black trees. In 1992, Adams used Join as a primitive for weight-balanced trees to implement set functions for union, intersection, and difference [1, 2]. Adams' algorithms were proposed in an international competition for the Standard ML community, which is about implementations on "set of integers". Adams won the "elegance" award in the competition, meaning that the algorithm is elegant yet reasonably efficient among all participants in the competition. However, Adams' original paper only informally argued a loose work bound, which is $O(m + n)$ for operating on two sets with sizes n and $m \leq n$.

Blelloch and Reid-Miller [10] first considered using join-based algorithms on parallel data structures. Their paper proposed parallel algorithms on treaps based on Join, and first show that these algorithms have $O(m \log(n/m))$ work and polylogarithmic span (with high probability). The algorithms are work-efficient because the work matches the lower bound of comparisons needed to combine two ordered sets. Later Blelloch et al. [7] studied the join-based algorithms, and showed work-efficiency and $O(\log n \log m)$ span for other balancing schemes, including AVL trees, read-black trees, and weight-balanced trees. In 2020 [9], the span bound for weight-balanced trees was improved to $O(\log n)$.

In 2015, Akhremtsev and Sanders [4] proposed join-based algorithms on (a, b)-trees. To make the algorithm more space-efficient and I/O-friendly in practice, Dhulipala et al. proposed C-trees [14] and PaC-trees [13]. The idea is to allow each tree node (or just the

leaves) to store a block of entries instead of one. Both C-trees and PaC-trees are binary. While their goal is to make the algorithms more cache-friendly in practice, no I/O bounds were shown. To the best of our knowledge, our work is the first to study the I/O cost of the join-based algorithms.

To achieve a better span bound, some techniques of our algorithms are inspired by existing work, such as using a list of pointers based on the height of subtrees [4], and using $\sqrt{m+n}$ way divide-and-conquer with a work-inefficient base case [9]. Our solution provides non-trivial and effective adaptions to use these ideas to achieve improved I/O bounds on B-trees.

Due to the elegance of the algorithm, many libraries later used the join-based algorithms for their implementation for sequences, sets, or maps (e.g., Haskell [19], the LEDA library [20], PAM [24], Aspen [14], CPAM [13]).

C B-Way Join Analysis

We will eventually show the I/O span of the B-Way-Join operation is $O(h_{\max} - h_{\min})$, and the I/O work is $O(B \cdot (h_{\max} - h_{\min}))$, where $h_{\max}$ and $h_{\min}$ are the maximum and minimal tree heights among the input trees. First, we must show that the height of the output tree of each recursive call does not increase the height of the input trees by more than a constant, as this could lead to an increase in the I/O work and span.

▶ **Lemma 5.** *Suppose we have an unbalanced B-tree T with b children and $b-1$ keys. Let C be the subset of children of T with height h, and the rest of the children have height $h-1$. Then we can rebalance the tree to have a valid B-tree of height h or $h+1$. If the tree is of height $h+1$, then the root will have at most $|C|$ keys.*

Proof. For each child $c \in C$, we can bring the keys up to the root node and reattach the children of c as children of the root node. Suppose c contains b' keys, then the number of keys at the root and the number of children both increase by b'.

The increase in the number of keys is straightforward. The number of children increases by the same amount because bringing c up adds $b'+1$ children, and we lost a child by bringing up c.

Performing this operation across all nodes in C will result in an at most $|C|B$ keys at the root. If the root node also contained B keys initially, then there would be $(|C|+1)B$ keys at the root, which would then be split into a tree of height $h+1$, and with at most $|C|$ keys at the root.

We know that the split will only ever occur once since $|C|$ is upperbounded to be at most B, thus a second split would not occur. ◀

▶ **Lemma 6.** *Suppose we have a list of trees $T_1, \ldots, T_{b+1}$ and keys $k_1, \ldots, k_b$, such that the height of each tree is either h and the root contains at most B keys, or $h+1$ and the root contains at most 1 key, except for T_1 which may contain 2 keys. Then we can join the trees such that the node at level $h+1$ contains at most $b + |H_i|$ keys, where H_i is a subset of trees $T_1, \ldots, T_i$ with height $h+1$. Note that if a tree is of height h, then the number of keys at the node of level $h+1$ is defined to be 0.*

Proof. We prove the lemma by induction for $i = 1, \ldots, b+1$. The base case trivially holds for $i = 1$, where if the height of the first tree is h, then the number of keys at the node of level $h+1$ is 0. If the height of the first tree is $h+1$, then the number of keys at the node of level $h+1$ is at most 2, maintaining the stated bound. We assume the claim is true for i, which means the first i trees produces B-tree T^* with at most $i - 1 + |H_i|$ keys at the node of level $h+1$.

If T^* is of height h, and the height of T_{i+1} is h, then the join will result in at most 1 key at level $h+1$, if T^* and T_{i+1} contain B keys each. In this case, T^* will be made the left child, T_{i+1} will be made the right child, and k_i will be the key at the root.

If T^* is of height $h+1$ and T_{i+1} is of height h, then the join will cause T_{i+1} to be the right child of T^*, and k_i will be the right most key of T^*. This increases the number of keys at the node of level $h+1$ by 1, and thus the bound is maintained. The same logic applies if T^* is of height h and T_{i+1} is of height $h+1$.

If instead both trees are of height $h+1$, then the join will cause the trees to be merged at the same height, and the number of keys increases by 1 plus the number of keys in T_{i+1}. Recall in the statement of the lemma that every tree except for T_1 contains at most 1 key at the root. Therefore, we can bound the increase by at most 2 keys, and we can see the bound is maintained since i increase by 1 and $|H_i|$ also increases by 1. ◀

▶ **Theorem 7.** *Let $T_1, \ldots, T_{b+1}$ and $k_1, \ldots, k_b$ be the set of input trees and keys respectively, where $b \leq B$. If h is the height of the tallest tree in T, then after performing* B-Way-Join*(T, k), the height of the output tree will be either h or $h+1$.*

Proof. We prove the theorem by induction. In the base case, if all trees in $T_{1,\ldots,b+1}$ are of height h, joining them maintains the theorem trivially. We create a new root node v containing all keys of k, and each tree in $T_{1,\ldots,b+1}$ is made a child of v. If any child node contains less than $B/2$ keys, then we can use a standard rebalancing algorithm to fix it. This may reduce the number of keys in v to less than b, but it will never increase. If all the keys in v are removed, then the output tree will also be of height h. Otherwise, it will be $h+1$.

Then we prove the inductive step. Let $T'_{0,1,\ldots}$ be the set of trees which contain children that are the result of a recursive call, i.e. the tallest trees in $T_{1,\ldots,b+1}$. Due to the inductive hypothesis, all the trees will have a right most child of height either $h-1$ or h. The first tree may also have a left most child of height h. Due to the possible discrepancy in heights, we must rebalance these trees. We know from Lemma 5, that we can rebalance these trees to have heights of h or $h+1$. If the height of a tree in T' is $h+1$, then we know there is at most 1 key at the root, except for T'_0 which may have 2 keys at the root.

According to Lemma 6, we can see that merging the trees in T' will result in a tree of height h or $h+1$. If the tree is of height h, then the proof is complete. Otherwise, we must ensure the root node has at most B keys. We know the root node has at most $b' + |H_{b'}|$ keys, where $b' = |T'|$. We know that $b' + |H_{b'}| \leq b$, since in order for a tree to have height $h+1$, it must have been passed into the recursive call with at least one other tree as part of the input set. Therefore, in order for $|H_{b'}|$ to increase by 1, the value of b' must have decreased by at least 1. Therefore, the number of keys at the root is upper bounded by b, and thus cannot overflow again, maintaining the height of the tree $h+1$. ◀

▶ **Theorem 4.1** (B-Way Join Analysis). *Let $T_1, T_2, \ldots, T_{b+1}$ be a set of B-trees, with the largest tree height $h_{\max}$ and the shortest tree height $h_{\min}$, and $k_1, k_2, \ldots, k_b$ be a set of separator keys, where $b \leq B$. The* B-Way-Join *operation can be performed in $O(h_{\max} - h_{\min})$ I/O span and $O(b \cdot (h_{\max} - h_{\min}))$ I/O work.*

Proof. We can see each separator key is assigned a thread, which is then responsible for traversing down a pair of input trees until it reaches the point where all trees are of the same height, which is a traversal of length of at most $O(h_{\max} - h_{\min})$.

At each level of the tree, each thread will perform at most $O(1)$ I/Os. We know from Theorem 7 at each level of the tree, the resulting subtree will always stay either the same height or increase by 1. Which means no additional I/Os must be performed in order to compensate for the increased height of the tree, resulting in an I/O span of $O(h_{\max} - h_{\min})$.

The I/O work comes readily from the I/O span. ◀

D Multiway Split: Cost Analysis

▶ **Theorem 1** (Multiway Split Cost). *We can split a B-tree T by $k_1, \ldots, k_d$ keys in parallel with $O(\log_B d + \log_B n)$ I/O span and $O(d \log_B n)$ I/O work, where n is $\sum_{i=1}^{d} |T_i|$, in the Fork-Join I/O Model.*

Proof. The algorithm spawns $d+1$ threads, where each thread performs a search down the tree. Each thread will perform $O(\log_B n)$ I/Os, in order to perform the search and copying the keys and pointers to the children takes $O(1)$ I/Os per node.

When performing the B-Way-Join operation, recall that the I/O span of the B-Way-Join operation is $O(h_{\max} - h_{\min})$, which is the difference in height between the tallest and shortest trees being joined together, and that the output tree must be of height at least $h_{\max}$.

We will be performing B-Way-Join along the path from the first divergence to each key. This means if at least one key is within the range of the split, then every B-Way-Join occurs with trees with height difference at most $O(1)$, and satisfying the I/O span bound of $O(\log_B n)$.

For the cases where a node contains keys within the range, but some of their ancestor's does not, we may use an amortized argument to charge a constant I/O for each node which does not contain keys within the range, which would otherwise not incur any cost. Then when joining with trees of height difference larger than a constant, we use the charge to pay for the extra I/O. The height difference can only be at most the number of consecutive skipped B-Way-Join operations.

For the I/O work bounds, observe that for each B-Way-Join operation, at most two trees being joined are of different heights, which are the trees along the search path to the split key. Therefore, the I/O work bounds is $O(1)$ instead of $O(B)$ for each B-Way-Join operation, which gives us the I/O work bound of $O(d \log_B n)$ I/Os. ◀

E Multiway Join Cost Analysis

▶ **Lemma 8.** *The cost of preprocessing d B-trees such that each tree T contains a list of pointers to all nodes along their left and right spines is $O(\log_B n + \log_B d)$ I/O span and $O(d \log_B n)$ I/O work, where n is the total number of keys in all trees.*

We can see that the fuse operations are completely independent of each other, and since each thread is able to access the number of keys that will be copied to the same tree ahead of them, then this process is able to occur in $O(1)$ I/O span and $O(B)$ I/O work, which gives us the following lemma.

▶ **Lemma 9.** *The cost of fusing B trees in the B-Way-Join-Fast operation is $O(1)$ I/O span and $O(B)$ I/O work.*

Next we show the following lemma, so that we can guarantee that after the first round of divisions, no overloaded node will push more than a single key up to their parent node.

▶ **Lemma 10.** *After fusing B trees together in the B-Way-Join-Fast operation, we can guarantee after one round of dividing the overloaded nodes, that each subsequent overloaded node may push up at most one key to their parent node.*

Proof. Recall that we are joining at most B trees together. Since the root and all nodes it may fuse with contains at most B keys, then including the paired key which the tree is associated with, each tree contributes at most 1 key to be pushed up to their parent node. In other words, for every tree that a node fuses with, they contribute at most 1 key to push to the parent per node.

Therefore, after a single round of divisions, even in the worst case where all keys were pushed up to a single node, this node would contain at most $2B$ keys, which would only push up a single key to the parent. ◀

▶ **Lemma 11.** *The cost of dividing the nodes up the tree in the B-Way-Join-Fast operation is $O(\log_2 \log_B n)$ I/O span and $O(B \log_B n)$ I/O work.*

Proof. The first set of divide operations cost $O(1)$ I/O span and $O(B)$ I/O work, since at most B new nodes are created, which can all be done in parallel.

Then we gather the number of keys in each node along the path from the root to every overloaded node. This takes $O(\log_B \log_B n)$ I/O span and $O(B \log_B n)$ I/O work, as we can use the list of pointers to jump directly to their assigned node along the path, since the path must consist of only nodes which were along the spines of a fused tree. There are at most B overloaded nodes, which means we create B arrays for the paths to the root.

We then perform the prefix sum operation along each of these arrays, which takes $O(\log_2 \log_B n)$ I/O span and $O(B \log_B n)$ I/O work. Finally, using the information provided by the prefix sum, each node can be divided in a $O(1)$ I/O for each thread, which gives us our desired results. ◀

▶ **Lemma 12.** *The cost of updating the list of pointers in the B-Way-Join-Fast operation is $O(\log_B \log_B n)$ I/O span and $O(B \log_B \log_B n)$ I/O work.*

Proof. Each list of pointers is a standard sequential B-tree. Therefore, the split operation takes $O(\log_B \log_B n)$ I/O span and work for each list. There are B lists of pointers, which gives us $O(B \log_B \log_B n)$ I/O work.

Then by using the B-Way-Join operation from Section 4.1, we can join these trees in $O(\log_B \log_B n)$ I/O span and $O(B \log_B \log_B n)$ I/O work. ◀

Now we can prove the main theorem of this section.

▶ **Theorem 13.** *We can join $T_1, \ldots, T_d$ B-trees and $k_1, \ldots, k_d$ keys together in parallel with $O(\log_B d \cdot \log_2 \log_B n + \log_B n)$ I/O span and $O(d \log_B n)$ I/O work, where n is $\sum_{i=1}^{d} |T_i|$, in the Fork-Join I/O Model.*

Proof. We get from Lemma 8 that the cost of preprocessing is $O(\log_B n + \log_B d)$ I/O span and $O(d \log_B n)$ I/O work.

The Multi-Join operation performs $O(\log_B d)$ rounds of B-Way-Join-Fast operations. This and preprocessing gives us the I/O span of $O(\log_B d \cdot \log_2 \log_B n + \log_B n)$.

To analyze the I/O work cost, we can see that the total number of B-Way-Join-Fast operations reduces by a factor of B each round. Therefore, we calculate the total I/O work cost from the following equation

$$\sum_{i=1}^{\log_B d} B^i \log_B n = O(d \log_B n).$$ ◀

F Prefix Sum

The prefix sums problem is a fundamental aspect of parallel computing. The applications for this problem extends to a wide variety of domains, including for the Parallel Joinable B-Tree.

▶ **Definition 14.** *Given an ordered set A of n elements, the prefix sums operation returns an ordered set A' of n elements, such that $B[i] = \sum_{j=0}^{i} A[j]$ for all $0 \leq i < n$.*

We show that if the input set A is located in contiguous main memory, then the prefix sums problem can be solved in the Fork-Join I/O Model in $O(\log_2 n)$ I/O span and $O(n)$ I/O work. By using the Gather operation, n non-contiguous elements can be gathered into a single contiguous array in $O(\log_B n)$ I/O span and $O(n)$ I/O work, thus the above result also holds for non-contiguous input sets.

▶ **Lemma 15.** *The prefix sum problem can be solved in the Fork-Join I/O Model in $O(\log_2 n)$ I/O I/O span and $O(n)$ I/O work.*

Proof. The Fork-Join I/O Model solution to the prefix sums problem is performed by simulating the optimal I/O work efficient PRAM algorithm. The algorithm performs two passes: building a tree bottom-up and then traversing the tree top-down. Each element in A is a leaf node, ordered from left to right, and we combine the elements in pairs to form a binary tree, where each node is the sum of its two children. Naturally this costs $O(\log_2 n)$ I/O span and $O(n)$ I/O work. Then from top down the tree, we pass the sum of the left child to the right child, summing up the left child's values until we reach the leaf, which then sums up the values of the left children passed. This top-down traversal also costs $O(\log_2 n)$ I/O span and $O(n)$ I/O work.

The tree can be implemented as an array, which would take advantage of the locality of the data. However, for the sake of simplicity, we will not use this optimization in our implementation as it does not affect the bounds of our algorithm. ◀

G Union Algorithm: Base Case Cost

▶ **Lemma 16.** *The Work-Inefficient-Union algorithm takes $O(\log_B m \cdot \log_2 \log_B n + \log_B n)$ I/O span and $O(m \log_B n)$ I/O work.*

Proof. The cost of the Work-Inefficient-Union algorithm is dominated by the Multi-Join operation, which comes readily from Theorem 13, where $d = m$. This gives us the I/O span $O(\log_B m \cdot \log_2 \log_B n + \log_B n)$ and I/O work $O(m \log_B n)$. ◀

▶ **Lemma 2** (Union Algorithm Base Case Cost)**.** *The Work-Inefficient-Union algorithm takes $O(\log_B m \cdot \log_2 \log_B n + \log_B n)$ I/O span and $O(m \log_B n)$ I/O work.*

Proof. Suppose we have k base cases, then the total cost of all base cases in the union algorithm is $\sum_{i=1}^{k} O(m_i(\log_B n_i + B))$, where m_i and n_i are the number of keys in the i-th base case. Recall that for each base case, $m_i < \sqrt{n_i + m_i}$.

From Lemma B.2 from Blelloch et al. [8], we know that $\sum_{i=1}^{k} m_i \log n_i = O(m \log n)$, where $\sum_{i=1}^{k} m_i = m, \sum_{i=1}^{k} n_i = n$, and for all i, $m_i \leq \sqrt{n_i + m_i}$. In the proof, the base of the logarithm is independent of the result, thus we can use the same proof to show that the I/O work is $O(m \log_B(\frac{n}{m}))$. ◀

H Additional Pseudocode for Multiway Split and Join

The pseudocode for the Multi-Split, Multi-Join, Join-With-Tall, and Divide-Node algorithms are ommitted from this version due to space constraints. However, they are provided in the full version of this paper. [2]

I Additional Set Operations

The Intersection and Difference operations are similar to the Union algorithm under the join-based framework. Hence, due to the space limit, we postpone the details of these two operations here in the appendix. The main difference lies in how subtrees are joined back.

▶ **Theorem 17.** *Given two B-trees with sizes m and $n \geq m$, there exists a parallel algorithm that returns a new B-tree containing the intersection and difference of the two input trees in $O\big(m \log_B\big(\frac{n}{m}\big)\big)$ I/O work, $O(\log_B m \cdot \log_2 \log_B n + \log_B n)$ and I/O span, where B is the block size.*

Proof. We first modify the Multi-Join operation slightly to take in an array d boolean values, where each boolean value indicates whether the corresponding key should be included in the final result. Suppose k_i is not to be part of the final result. Then we instead find the largest key in T_i, denoted as k_i', and split on T_i using k_i'. Then we replace k_i with k_i', and perform the join operation as normal.

In the Intersection operation, after searching for the $d-1$ split keys, we check if all $d-1$ keys are in both trees. If a key is in both trees, then we mark the key to be included in the final result. Then we perform the Multi-Split operation as normal, make our recursive calls, then use the modified Multi-Join operation to combine the results, excluding any keys which did not appear in both trees.

The base case of the Intersection operation is also very similar to the Work-Inefficient-Union algorithm. Let $|T_1| \geq |T_2|$, as we can always swap the two trees if this is not the case. Then we use the keys from T_2 to search for the keys in T_1. Any keys which are not found in T_1 are discarded. All remaining keys are then joined together using the traditional Multi-Join operation.

For the Difference operation, if a split key is found to be in T_1 and T_2 during the main recursive algorithm, then it is marked as excluded, and all the same steps as the Intersection operation are performed. For the base case, we use a slightly different approach depending on which tree is larger. If $|T_1| \geq |T_2|$, then we split T_1 using the keys in T_2, and join with none of the keys from T_2. Otherwise, we use all keys in T_1 to search for the keys in T_2, and join all keys from T_1 which were not found in T_2.

As such, the same analysis used to get the bounds from the Union algorithm can also be applied to the Intersection and Difference operations and achieve the same bounds . ◀

[2] The full version of this paper is available at `https://arxiv.org/abs/2510.20053`.

Structural Parameters for Steiner Orientation

Tesshu Hanaka
Kyushu University, Fukuoka, Japan

Michael Lampis
Université Paris-Dauphine, PSL University, CNRS UMR7243, LAMSADE, Paris, France

Nikolaos Melissinos
Computer Science Institute, Faculty of Mathematics and Physics,
Charles University, Prague, Czech Republic

Edouard Nemery
Université Paris-Dauphine, PSL University, CNRS UMR7243, LAMSADE, Paris, France

Hirotaka Ono
Nagoya University, Japan

Manolis Vasilakis
Université Paris-Dauphine, PSL University, CNRS UMR7243, LAMSADE, Paris, France

Abstract

We consider the Steiner Orientation problem, where we are given as input a mixed graph $G = (V, E, A)$ and a set of k demand pairs (s_i, t_i), $i \in [k]$. The goal is to orient the undirected edges of G in a way that the resulting directed graph has a directed path from s_i to t_i for all $i \in [k]$. We adopt the point of view of structural parameterized complexity and investigate the complexity of Steiner Orientation for standard measures, such as treewidth. Our results indicate that Steiner Orientation is a surprisingly hard problem from this point of view. In particular, our main contributions are the following:

1. We show that Steiner Orientation is NP-complete on instances where the underlying graph has feedback vertex number 2, treewidth 2, pathwidth 3, and vertex integrity 6.
2. We present an XP algorithm parameterized by vertex cover number vc of complexity $n^{\mathcal{O}(\mathrm{vc}^2)}$. Furthermore, we show that this running time is essentially optimal by proving that a running time of $n^{o(\mathrm{vc}^2)}$ would refute the ETH.
3. We consider parameterizations by the number of undirected or directed edges ($|E|$ or $|A|$) and we observe that the trivial $2^{|E|}n^{\mathcal{O}(1)}$-time algorithm for the former parameter is optimal under the SETH. Complementing this, we show that the problem admits a $2^{\mathcal{O}(|A|)}n^{\mathcal{O}(1)}$-time algorithm.

In addition to the above, we consider the complexity of Steiner Orientation parameterized by $\mathrm{tw} + k$ (FPT), distance to clique (FPT), and $\mathrm{vc} + k$ (FPT with a polynomial kernel).

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases ETH, Steiner Orientation, Treewidth

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.38

Related Version *Full Version*: `https://arxiv.org/abs/2507.21445`

Funding *Tesshu Hanaka*: Partially supported by JSPS KAKENHI Grant Numbers JP21K17707, JP22H00513, JP23H04388, JP25K03077, and JST, CRONOS, Japan Grant Number JPMJCS24K2.
Michael Lampis: Partially supported by ANR project ANR-21-CE48-0022 (S-EX-AP-PE-AL).
Nikolaos Melissinos: Partially supported by Charles University projects UNCE 24/SCI/008 and PRIMUS 24/SCI/012, and by the project 25-17221S of GAČR.
Hirotaka Ono: Partially supported by JSPS KAKENHI Grant Numbers JP20H05967, JP22H00513, JP24K02898, JP25K03077, and JST, CRONOS, Japan Grant Number JPMJCS24K2.
Manolis Vasilakis: Partially supported by ANR project ANR-21-CE48-0022 (S-EX-AP-PE-AL).

© Tesshu Hanaka, Michael Lampis, Nikolaos Melissinos, Edouard Nemery, Hirotaka Ono, and Manolis Vasilakis;
licensed under Creative Commons License CC-BY 4.0

1 Introduction

The Steiner Orientation problem is an NP-hard graph optimization problem, which involves assigning directions to the undirected edges of a mixed graph – a graph containing both directed and undirected edges – to satisfy specific connectivity requirements, given by pairs of terminals. The objective is to orient the undirected edges in such a way that there exists a directed path between each given terminal pair. Formally, the problem is defined as follows.

Steiner Orientation

Instance: A mixed graph $G = (V, E, A)$ where E and A denote the set of edges and arcs in G respectively. Additionally, we are given a set of terminal pairs $\mathcal{T} = \{(s_i, t_i) \in V \times V : i \in [k]\}$ such that $s_i \neq t_i$, for all $i \in [k]$.

Goal: Determine whether there exists an orientation of E such that k directed paths $P_i = s_i \to \ldots \to t_i$, $i \in [k]$ exist.

The Steiner Orientation problem has been well-studied in bioinformatics, motivated by modeling protein–protein or protein–DNA interactions [9, 16, 18, 19]. Another motivation naturally arises from designing transportation networks. For example, in an urban transportation network with a mix of one-way streets (directed arcs) and two-way streets (undirected edges), one can consider the traffic control problem of deciding the direction of some streets to ensure routes from specific origins to destinations. This scenario is naturally modeled as the Steiner Orientation problem.

Regarding its complexity, Arkin and Hassin show that Steiner Orientation is NP-complete in general, but polynomially solvable if $k = 2$ [1]. From the perspective of parameterized complexity, Cygan et al. [7] proposed an $n^{\mathcal{O}(k)}$-time algorithm for the number of terminal pairs k as a parameter, which shows that the problem belongs to the class XP. However, subsequent research has revealed the problem's intractability for this parameterization; Steiner Orientation is shown to be W[1]-hard when parameterized by k and cannot be solved in time $f(k) \cdot n^{o(k/\log k)}$ under ETH [17]. This lower bound was later improved to a $f(k) \cdot n^{o(k)}$ ETH-based lower bound even on planar graphs [4]. Surprisingly, Włodarczyk recently proved that the problem is not just W[1]-hard, but W[1]-complete [20]. This series of studies has almost completely characterized the complexity with respect to the parameter k. Despite this literature, to the best of our knowledge, the parameterized complexity of Steiner Orientation is only studied for the number k of terminal pairs. The complexity with respect to graph parameters such as treewidth or vertex cover number has been almost entirely unstudied. This paper aims to provide the first systematic study in this direction.

1.1 Our Contribution

In this paper, we first show the para-NP-completeness of Steiner Orientation for the parameters pathwidth, feedback vertex number, and vertex integrity.[1] Specifically, the problem is NP-complete on series-parallel graphs (i.e., graphs of treewidth at most 2) of vertex-deletion distance 2 to a forest of stars of size at most 4. By slightly modifying our reduction, we further obtain NP-hardness for grid graphs (which are planar and have

[1] In this paper, graph parameters refer to those of the underlying graph of G, that is, the graph obtained by replacing all arcs with undirected edges. We assume that the reader is familiar with standard parameters, but recall some of the relevant definitions and relations further below.

maximum degree 4) with bounded pathwidth. Notice that, in a sense, these results clearly delineate the frontier of polynomial-time versus NP-complete cases for most standard graph parameters: On the one hand, when we place no bound on the degree, Steiner Orientation is already NP-hard on graphs which are 2 vertices away from an extremely restricted class. On the other hand, for bounded-degree graphs no such result can be obtained, as any graph that has bounded tree-depth and bounded degree actually has bounded size, so the hardness for bounded pathwidth graphs is essentially best possible.

The results above motivate us to consider more restrictive parameters. In this direction, we present an XP algorithm parameterized by vertex cover number, vc, that runs in time $n^{\mathcal{O}(\mathrm{vc}^2)}$. Even though this improves the situation compared to the paraNP-completeness for treewidth, this complexity is rather disappointing, raising the natural question whether the square in the exponent is necessary. Our main contribution for this parameter is to answer this question by proving that an $n^{o(\mathrm{vc}^2)}$ time algorithm would refute the ETH. Along the way, we also show that Steiner Orientation is fixed-parameter tractable when parameterized by distance to a clique, which can be seen as the dense analogue of vertex cover (a vertex cover of a graph is a clique modulator of its complement).

Moving in another direction, we consider the number of undirected or directed edges as parameters. Here, we show that Steiner Orientation is fixed-parameter tractable (FPT) when parameterized by the number $|A|$ of directed edges. We observe that if $|A| = 0$, the problem is solvable in polynomial time as a consequence of Robbins' theorem [14]. Thus, our result generalizes this classical polynomial-time case. Here, it is natural to consider the number $|E|$ of undirected edges. Since we have only two directions for each undirected edge, Steiner Orientation is clearly solvable in $2^{|E|}n^{\mathcal{O}(1)}$ time. Indeed, this running time is essentially tight, as a careful observation of the standard NP-hardness reduction for Steiner Orientation [1] yields a $(2-\varepsilon)^{|E|}n^{\mathcal{O}(1)}$ lower bound under the SETH.

Finally, we investigate the parameterized complexity with respect to the combined parameters $k+\mathrm{tw}$ and $k+\mathrm{vc}$. We show that Steiner Orientation is FPT when parameterized by $k+\mathrm{tw}$ by formulating it as an MSO_2 problem, which stands in contrast to the NP-hardness on series-parallel graphs. Furthermore, we give a polynomial kernel when parameterized by $k+\mathrm{vc}$.

Overview of Techniques. Our NP-completeness proof for Steiner Orientation on graphs of treewidth 2 is based on a direct reduction from a variant of 3-SAT, where we naturally use undirected edges to represent variables of the initial formula, with directions representing truth assignments (this is standard). The key new insight is the following: we can transmit information between copies of such edges, ensuring they must be oriented in a consistent way, by adding demand pairs which must be routed through *two* main hub vertices. By using appropriately oriented arcs to connect our gadgets to the two hubs we ensure that gadgets don't interfere with each other. In a sense, the take-home message of this construction is that the structure of G alone is not enough to measure the complexity of an instance, because the way that demand pairs interact with the graph can significantly complicate the structure of the instance.

The XP algorithm parameterized by vertex cover is based on a simple branching procedure: for each pair u, v of vertices of the vertex cover, guess if there is a directed path of length at most 2 from u to v in the solution and, if yes, which is (potentially) the vertex w in the middle of the path. This clearly leads to at most $n^{\mathcal{O}(\mathrm{vc}^2)}$ guesses, and once we have fixed these decisions it is not hard to complete the solution because we can infer exactly which vertices of the cover have directed paths to each other.

Given the simplicity of the algorithm sketched above for vertex cover, it is somewhat surprising that this is essentially optimal (under the ETH). We establish this by giving a reduction from k-Clique producing a Steiner Orientation instance with vertex cover $\mathcal{O}(\sqrt{k})$. The key intuition is again that we can use a small number ($\mathcal{O}(\sqrt{k})$) of hubs, through which information will be routed. In particular, we set up a complete bipartite graph $K_{\sqrt{k},\sqrt{k}}$ and for each edge of this graph we set up a gadget encoding a selection in the original instance: the number of edges of this graph is sufficient to encode all selections, but the vertex cover of the construction is determined by the number of vertices of the complete bipartite graph.

To obtain an algorithm for parameter $|A|$ we face significantly more obstacles than for parameter $|E|$ (which is trivially FPT). Our approach relies on exhaustive applications of some (standard) reduction rules for this problem, which eliminate all cycles and degree 1 vertices. We then focus on a restricted special case of Steiner Orientation, where all undirected components are paths, all internal vertices of the paths are incident on no arcs, and the endpoints of each path are incident on at most one arc. We solve this restricted case using a reduction to 2-SAT. We then show, using arguments which essentially take into account that A is a feedback edge set of the underlying graph, that a simple branching algorithm can reduce any instance into $2^{\mathcal{O}(|A|)}$ instances of the (polynomial-time solvable) restricted case.

The FPT algorithm for distance to clique relies again on exhaustively applying reduction rules that eliminate cycles and then making some simple structural observations: there are few edges incident on the modulator vertices (so we can guess their orientation); and the edges contained in the clique must have a very simple structure (they form a matching). The FPT algorithm for $\mathrm{tw} + k$ relies on a formulation of the problem in MSO_2 logic and Courcelle's theorem – we leave it as an interesting open problem to determine the best dependence for this parameterization. Finally, for the more restrictive parameter $\mathrm{vc} + k$ we obtain a polynomial kernel via a maximum matching argument. In particular, we show that by calculating a maximum matching between pairs of vertices of the vertex cover and non-terminals in the indpendent set, we can identify a set of $\mathcal{O}(\mathrm{vc}^2)$ vertices of the independent set which are sufficient to obtain a solution and delete the rest, giving a kernel of order $\mathcal{O}(\mathrm{vc}^2 + k)$.

1.2 Related work

The Steiner Orientation problem has been studied extensively. From a bioinformatics perspective, an optimization version of Steiner Orientation is widely studied [9,16,18,19]. In this optimization version, called Maximum Steiner Orientation, the goal is to find an orientation that maximizes the number of satisfied terminals pairs.

Unfortunately, Maximum Steiner Orientation is significantly more difficult than Steiner Orientation. Medvedovsky et al. [16] show that the problem is inapproximable within a factor of 12/13 even on undirected stars and binary trees. Elberfeld et al. then observe that the NP-hardness reduction for Steiner Orientation in [1] implies the inapproximability within a factor of 7/8 [10]. More recently, Hörsch [15] considered the maximization version of Steiner Orientation but under the restriction that the set of demand pairs includes all possible demands and showed that this case is also APX-hard. Interestingly, for this version of the problem Hörsch supplies an XP algorithm for parameter $|A|$ and leaves it as an open question whether an FPT algorithm can be obtained. In this paper we provide an FPT algorithm for Steiner Orientation parameterized by $|A|$, but this is somewhat orthogonal to Hörsch's question, as we are not dealing with the maximization version (which makes our problem easier) but we are also not assuming that all pairs of vertices have a demand (which makes our problem harder).

For the approximability, an $\mathcal{O}(\log n)$-approximation is presented for undirected graphs [16]. This has been improved to a factor $\mathcal{O}(\log n/\log\log n)$ [11], and later to $\mathcal{O}(\log k/\log\log k)$ [7]. For bounded feedback vertex number graphs and bounded treewidth graphs, an $\mathcal{O}(\log n)$-approximation algorithm and an $\mathcal{O}(\log^2 n)$-approximation algorithm are presented [10].

From the viewpoint of parameterized complexity, Dorn et al. [9] and Roayaei [18] propose several FPT algorithms for parameters related to terminal pairs.

Regarding the parameterized approximation, Chitnis, Feldmann, and Suchý [4] show that there is no constant $\epsilon > 0$ such that Maximum Steiner Orientation on planar graphs admits an $(19/20-\epsilon)$-approximation[2] in time $f(k)n^{o(1)}$. Włodarczyk [20] shows that there is no $(\log k)^{o(1)}$-approximation algorithm for Maximum Steiner Orientation that runs in FPT time with respect to k assuming FPT $\neq$ W[1].

2 Preliminaries

We use the standard notations in graph theory. Let $G = (V, E, A)$ be a mixed graph where V is a set of vertices, E is a set of undirected edges, and A is a set of directed edges. We denote by $\{u, v\} \in E$ an undirected edge between u and v and by $(u, v) \in A$ a directed edge from u to v.

A mixed graph G is a *mixed acyclic graph* if it has no cycle. Notice that in the context of mixed graphs a cycle is any sequence of vertices $v_1, v_2, \ldots, v_p$ with $v_1 = v_p$ such that between any two consecutive vertices v_i, v_{i+1} the graph contains either the edge $\{v_i, v_{i+1}\}$ or the arc (v_i, v_{i+1}). By definition, the subgraph induced by undirected edges in a mixed acyclic graph is a forest. Moreover, the graph obtained from a mixed acyclic graph G by contracting all the undirected edges is a directed acyclic graph. For a vertex subset $V' \subseteq V$, we denote by $G[V'] = (V', E(V'), A(V'))$ the subgraph induced by V', where $E(V') \subseteq E$ and $A(V') \subseteq A$. A mixed path of a mixed graph is a path using both edges and arcs respecting the orientation of the arcs. For a mixed graph $G = (V, E, A)$, $\lambda(E)$ denotes an orientation of E and G_λ denotes the directed graph obtained by the orientation $\lambda(E)$.

Graph parameters. Throughout the paper we will use several structural graph parameters which will in general refer to the underlying graph of a given mixed graph G. Recall that the underlying graph is the graph obtained by replacing each arc by an edge with the same endpoints. The parameters we will mention are treewidth (tw), pathwidth (pw), vertex integrity (vi), feedback vertex set (fvs), vertex cover (vc), and distance to clique (dtc). For the definitions of treewidth and pathwidth we refer the reader to [6]; fvs, vc, and dtc denote the size of the smallest set of vertices whose removal leaves the graph a forest, an independent set, or a clique respectively; while a graph has vertex integrity at most k if there exists a set of vertices S whose removal results in a graph where all components have size at most $k - |S|$. It is known that four of these parameters form a hierarchy, in the sense that $\text{tw} \leq \text{pw} \leq \text{vi} \leq \text{vc} + 1$ for all graphs. We note that all these parameters are closed under vertex deletion and edge contraction, meaning that these operations can only decrease the parameter value of a graph.

[2] Note that in [4] the approximation ratio is defined as the value of the approximate solution divided by the value of the optimal solution, which results in ratios smaller than 1. In most other works, the ratio is instead defined as the optimal solution divided by the approximate solution, which gives values larger than 1.

2.1 Preprocessing

We present two basic polynomial-time reduction rules for Steiner Orientation, which eliminate cycles and degree-1-vertices respectively. Applying these rules will never increase the values of our parameters, as the rules use edge contractions and vertex deletions, so in the rest of the paper we will mostly focus on instances where these rules have been exhaustively applied.

▶ **Proposition 1** ([19]). *Let $(G, \mathcal{T})$ be an instance of* Steiner Orientation. *Let G' be a mixed graph obtained from G by contracting all the cycles and $\mathcal{T}'$ be the corresponding set of terminal pairs. Then $(G, \mathcal{T})$ is a yes-instance if and only if $(G', \mathcal{T}')$ is a yes-instance. Moreover, G' is a mixed acyclic graph and can be computed in polynomial time.*

▶ **Proposition 2.** *Let $(G, \mathcal{T})$ be an instance of* Steiner Orientation. *If G has a vertex $v \in V$ of degree 1 in the underlying graph, then we can either correctly conclude that $(G, \mathcal{T})$ is a no-instance or construct a new instance $(G', \mathcal{T}')$ such that $G' = G[V \setminus \{v\}]$ and $(G, \mathcal{T})$ is a yes-instance if and only if $(G', \mathcal{T}')$ is a yes-instance.*

3 NP-hardness

Our main result in this section is to show that Steiner Orientation remains NP-hard even on very restricted families of graphs. We focus on restrictions on the structure of the underlying undirected graph and show that even if the underlying graph is series-parallel and has vertex-deletion distance 2 to a forest of stars of size at most 4 the problem is still NP-complete. Note that all series-parallel graphs have treewidth at most 2 [2, Theorem 41] and are planar [3, Chapter 11.2], while the graph constructed by our reduction has feedback vertex number 2, pathwidth 3, and vertex integrity 6. Since Steiner Orientation is polynomial-time solvable when the underlying graph of the input is a tree (as there exists a unique path between every pair of terminals), our result establishes a sharp dichotomy on the polynomial-time solvability of Steiner Orientation with respect to the treewidth of the input graph. Moreover, by slightly modifying our reduction we further obtain NP-hardness for grid graphs of bounded pathwidth.

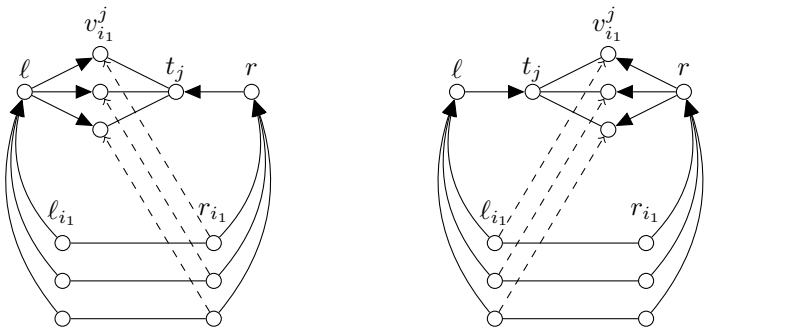
▶ **Theorem 3.** Steiner Orientation *is NP-complete on series-parallel graphs of vertex-deletion distance 2 to a forest of stars of size at most 4.*

Proof. It is easy to see that Steiner Orientation belongs to NP, thus in the rest of the proof we argue about its NP-hardness. The starting point of our reduction is the Monotone 3-SAT problem, which is the variation of 3-SAT where every clause contains at most 3 literals and is *monotone*, that is, it contains only unnegated or only negated variables. This variation is well-known to be NP-hard [8, 12].

Let ϕ be an instance of Monotone 3-SAT, where $X = \{x_1, \ldots, x_n\}$ denotes its variables and $C = \{c_1, \ldots, c_m\}$ its clauses. We say that a clause is *positive* if it consists of only unnegated variables, and *negative* if it consists of only negated variables.

We will construct an instance $(G, \mathcal{T})$ of Steiner Orientation that is equivalent to ϕ. For each variable x_i, introduce vertices ℓ_i, r_i that are connected via an undirected edge. We further introduce vertices ℓ and r, as well as the arcs $\ell_i \to \ell$ and $r_i \to r$, for all $i \in [n]$. Now let c_j be a positive clause containing 3 variables, that is, $c_j = x_{i_1} \vee x_{i_2} \vee x_{i_3}$ for distinct $i_1, i_2, i_3 \in [n]$. We construct the corresponding clause gadget as depicted in Figure 1. In particular, we introduce vertices $v^j_{i_1}, v^j_{i_2}, v^j_{i_3}$, all of which have an incoming arc from ℓ, as well as a vertex t_j that is connected via undirected edges with $v^j_{i_1}, v^j_{i_2}, v^j_{i_3}$ and has an incoming arc

from r. We further introduce the terminal pairs $(\ell, t_j), (r_{i_1}, v^j_{i_1}), (r_{i_2}, v^j_{i_2}), (r_{i_3}, v^j_{i_3})$ into $\mathcal{T}$. We treat positive clauses that contain 2 variables in an analogous way, while the construction for the negative clauses is entirely symmetric.



(a) Positive clause $c_j = x_{i_1} \vee x_{i_2} \vee x_{i_3}$. **(b)** Negative clause $c_j = \neg x_{i_1} \vee \neg x_{i_2} \vee \neg x_{i_3}$.

Figure 1 The terminal pairs added during the construction of the clause gadget are denoted by dashed arcs, apart from (ℓ, t_j) and (r, t_j) respectively for the sake of clarity.

This completes the construction of the instance $(G, \mathcal{T})$ of Steiner Orientation. It is easy to see that G is series-parallel, while $G - \{\ell, r\}$ is a forest whose connected components are either a K_2, a $K_{1,2}$, or a $K_{1,3}$, thus indeed it holds that $\mathrm{fvs}(G) = 2$, $\mathrm{pw}(G) = 3$, and $\mathrm{vi}(G) = 6$. It remains to argue that the instance is equivalent to ϕ.

For the forward direction, let $\alpha\colon X \to \{\mathsf{true}, \mathsf{false}\}$ be a satisfying assignment for ϕ. Consider the following orientation for the undirected edges of G. If $\alpha(x_i) = \mathsf{true}$ then we orient the edge $\{\ell_i, r_i\}$ as $r_i \to \ell_i$, otherwise if $\alpha(x_i) = \mathsf{false}$ then we orient it as $\ell_i \to r_i$. As for the clauses c_j that contain x_i and the edge $\{v^j_i, t_j\}$, if c_j is a positive clause (resp. negative) then if $\alpha(x_i) = \mathsf{true}$ we orient it as $v^j_i \to t_j$ (resp. $t_j \to v^j_i$), otherwise if $\alpha(x_i) = \mathsf{false}$ we orient it as $t_j \to v^j_i$ (resp. $v^j_i \to t_j$). It suffices to argue that this orientation satisfies all terminal pairs in $\mathcal{T}$.

Let $c_j = x_{i_1} \vee x_{i_2} \vee x_{i_3}$ be a positive clause involving three variables. We will argue that the described orientation satisfies all terminal pairs introduced due to the clause gadget of c_j. Since α is a satisfying assignment for ϕ, it holds that there exists $i \in \{i_1, i_2, i_3\}$ such that $\alpha(x_i) = \mathsf{true}$. Consequently, the arc $v^j_i \to t_j$ exists in our orientation, thus the terminal pair (ℓ, t_j) is satisfied by the directed path $\ell \to v^j_i \to t_j$. As for the terminal pair $(r_{i'}, v^j_{i'})$, where $i' \in \{i_1, i_2, i_3\}$, notice that if $\alpha(x_{i'}) = \mathsf{true}$ then it is satisfied by the directed path $r_{i'} \to \ell_{i'} \to \ell \to v^j_{i'}$, while if $\alpha(x_{i'}) = \mathsf{false}$ then it is satisfied by the directed path $r_{i'} \to r \to t_j \to v^j_{i'}$. The argumentation is similar for positive clauses involving two variables, and symmetric for negative clauses.

For the converse direction, assume there exists an orientation of G such that all terminal pairs in $\mathcal{T}$ are satisfied. Consider the assignment $\alpha\colon X \to \{\mathsf{true}, \mathsf{false}\}$ where $\alpha(x_i) = \mathsf{true}$ if and only if the edge $\{\ell_i, r_i\}$ has been oriented as $r_i \to \ell_i$. We will prove by contradiction that this is a satisfying assignment for ϕ. Let $c_j = x_{i_1} \vee x_{i_2} \vee x_{i_3}$ be a positive clause involving three variables, and assume that $\alpha(x_i) = \mathsf{false}$ for all $i \in \{i_1, i_2, i_3\}$. In that case, for the terminal pair (r_i, v^j_i) to be satisfied, it follows that the edge $\{v^j_i, t_j\}$ has been directed as $t_j \to v^j_i$. Since this holds for all $i \in \{i_1, i_2, i_3\}$, it contradicts the fact that the terminal pair (ℓ, t_j) is satisfied by the orientation. The argumentation is similar for positive clauses involving two variables, and symmetric for negative clauses. ◀

▶ **Corollary 4.** Steiner Orientation *is NP-complete on grid graphs of constant pathwidth.*

4 FPT algorithm by distance to clique

In this section, we give an FPT algorithm parameterized by distance to clique. More precisely, we consider the case where the underlying undirected graph of the input is at most dtc vertex deletions away from being a clique. In a sense, this parameterization is the dense analogue of the parameterization by vertex cover (because a vertex cover of a graph is a clique modulator of its complement). As usual, we will assume that the set whose removal leaves the underlying graph a clique is given to us in the input. The main result of this section is the following.

▶ **Theorem 5.** Steiner Orientation *can be solved in time* $4^{\mathrm{dtc}}n^{\mathcal{O}(1)}$.

We establish Theorem 5 starting via a series of reduction rules given by the lemmas below. Throughout we will assume that we are given a set S of dtc vertices such that removing S from the input graph leaves a mixed graph whose underlying graph is a clique. We will also assume that, using Proposition 1, the input is a mixed *acyclic* graph.

▶ **Lemma 6.** *Let G be a mixed acyclic graph and S a set of vertices such that the underlying graph of $G - S$ is a clique. Then, each vertex in S has at most one undirected edge to $V \setminus S$.*

▶ **Lemma 7.** *In a mixed acyclic graph G, let C be a clique in the underlying graph of G. Then there is no vertex incident to at least two undirected edges in C. In other words, the set of undirected edges in C forms a matching.*

We can now proceed to the proof of Theorem 5. The high-level idea is that by using our previous observations the number of undirected edges incident on S is at most 2dtc; we will therefore enumerate all possible orientations of these edges and check for each orientation if it can be extended to a feasible solution. The challenge will be in establishing that this check can be performed in polynomial time.

5 Vertex cover number

In this section we consider Steiner Orientation parameterized by the vertex cover number vc of the input graph, arguably one of the most restrictive structural parameterizations one could consider. We first show in Theorem 8 that even in this extremely restrictive setting, the problem remains intractable and prove its W[1]-hardness. As a matter of fact, our reduction implies that the problem does not admit any $n^{o(\mathrm{vc}^2)}$-time algorithm under the ETH, and our next result is in Theorem 10 to present such an optimal algorithm.

5.1 Hardness

▶ **Theorem 8.** Steiner Orientation *is W[1]-hard parameterized by the vertex cover number* vc *of the input graph, and for any computable function f it cannot be solved in time $f(\mathrm{vc})n^{o(\mathrm{vc}^2)}$ under the ETH.*

Proof. In k-Multicolored Clique we are given a graph G and a partition of $V(G)$ into k independent sets each of size n, and we are asked to determine whether G contains a k-clique. It is well-known that k-Multicolored Clique parameterized by k is W[1]-hard and does not admit any $f(k)n^{o(k)}$-time algorithm, where f is any computable function, unless the ETH is false [6]. Let (G, k) be an instance of k-Multicolored Clique, where we assume without loss of generality that $\sqrt{k} \in \mathbb{N}$ (one can do so by adding dummy independent sets connected to all the other vertices of the graph). Recall that we assume that G is given to us partitioned into k independent sets $V_1, \ldots V_k$, where $V_i = \{v_1^i, \ldots, v_n^i\}$. We will construct in polynomial time an equivalent instance $(H, \mathcal{T})$ of Steiner Orientation, with $\mathrm{vc}(H) = \mathcal{O}(\sqrt{k})$.

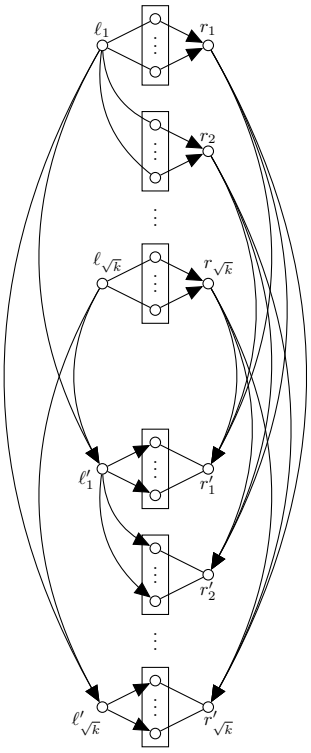


Figure 2 Part of the construction of graph H. Rectangles denote independent sets of size n.

Construction. We first introduce the vertices $\{\ell_\alpha, \ell'_\alpha, r_\alpha, r'_\alpha : \alpha \in [\sqrt{k}]\}$ and add the arcs $\ell_\alpha \to \ell'_\beta$ and $r_\alpha \to r'_\beta$ for all $\alpha, \beta \in [\sqrt{k}]$. For all $i \in [k]$, we further introduce the independent sets $X_i = \{x_1^i, \ldots, x_n^i\}$ and $Y_i = \{y_1^i, \ldots, y_n^i\}$. We fix a bijective function $h\colon [k] \to [\sqrt{k}]^2$ that maps every independent set V_i to a distinct pair $(\alpha, \beta) \in [\sqrt{k}]^2$, and for all $i \in [k]$ with $h(i) = (\alpha, \beta)$

- we add the edge $\{\ell_\alpha, x_j^i\}$ for all $x_j^i \in X_i$,
- we add the arc (x_j^i, r_β) for all $x_j^i \in X_i$,
- we add the arc (ℓ'_α, y_j^i) for all $y_j^i \in Y_i$,
- we add the edge $\{r'_\beta, y_j^i\}$ for all $y_j^i \in Y_i$.

This completes the construction of the graph H, and we refer to Figure 2 for an illustration of a part of it. As for the terminal pairs, we proceed in three steps:

1. We initially add into $\mathcal{T}$ the pairs (ℓ_α, r_β) and (ℓ'_α, r'_β), for all $\alpha, \beta \in [\sqrt{k}]$.
2. Next, for all $i \in [k]$ and $j \in [n]$, we add the terminal pair $(x_j^i, y_{j'}^i)$ into $\mathcal{T}$, for all $j' \in [n] \setminus \{j\}$. We refer to the terminal pairs added in this step as *consistency terminal pairs*.
3. Lastly, for distinct $i_1, i_2 \in [k]$, if $\{v_{j_1}^{i_1}, v_{j_2}^{i_2}\} \notin E(G)$ we add the terminal pairs $(x_{j_1}^{i_1}, y_{j_2}^{i_2}), (x_{j_2}^{i_2}, y_{j_1}^{i_1})$ into $\mathcal{T}$, where $j_1, j_2 \in [n]$. We refer to the terminal pairs added in this step as *edge-checking terminal pairs*.

This completes the construction of the instance $(H, \mathcal{T})$.

It is easy to see that $\{\ell_\alpha, \ell'_\alpha, r_\alpha, r'_\alpha : \alpha \in [\sqrt{k}]\}$ is a vertex cover of H, thus $\mathrm{vc}(H) = \mathcal{O}(\sqrt{k})$. To complete the proof, we show the following claim.

▷ Claim 9. The instance (G, k) of k-Multicolored Clique is a yes-instance if and only if the constructed instance $(H, \mathcal{T})$ of Steiner Orientation is a yes-instance. ◀

5.2 XP algorithm

In this section, we give an XP algorithm parameterized by vc with complexity $n^{\mathcal{O}(\text{vc}^2)}$, matching our lower bound result.

▶ **Theorem 10.** *There is an algorithm that solves* Steiner Orientation *in time* $n^{\mathcal{O}(\text{vc}^2)}$.

Proof. First we note that contracting all mixed cycles as in Proposition 1 does not increase the size of a minimal vertex cover, so we can suppose that our graph is mixed acyclic.

Let G be a graph of vertex cover number vc. Let $S \subseteq V$ be a vertex cover G of size vc. It can be computed in time $2^{\mathcal{O}(\text{vc})}$ [13]. First we guess the orientation of the edges of $G[S]$. There are $2^{\mathcal{O}(\text{vc}^2)}$ possibilities. Then, for all pairs $(u, v) \in S^2$, we guess if there is, in the final solution, an oriented path $u \to w \to v$ with $w \in V \setminus S$; if we guessed there is at least one, we guess one such w, and orient its edges in consequence. There are $n^{\mathcal{O}(\text{vc}^2)}$ possibilities. As we will see in Claim 12, after this guessing phase, as S is a vertex cover, the connectivity of S in any solution respecting the guesses is already decided. This will allow us to check the existence of such a solution in polynomial time.

▷ Claim 11. Let $w \in V \setminus S$, $u, v \in S$ such that at the end of the guessing steps, there are two undirected edges $\{u, w\}$ and $\{v, w\}$. In any feasible solution respecting the guesses, either we orient them both from w to u, v or from u, v to w.

The above claim shows that for any $w \in V \setminus S$, after the guessing steps, either all of its undirected edges must be oriented from w to S or from S to w.

▷ Claim 12. Let $u, v \in V$; in any solution respecting the guesses, if there is a path from u to v, then there is one that only uses already oriented edges after the guessing steps, except possibly for the first one if $u \in V \setminus S$, and the last one if $v \in V \setminus S$.

This claim shows us that after the guessing steps, for any (s_i, t_i) pair of terminals that is not already satisfied, the only way to satisfy them while respecting the guessing steps is to orient the edges of s_i and t_i.

We now argue that there exists a demand pair (s_i, t_i) which should be treated "first". To see this, consider an auxiliary directed graph which has a vertex for each demand pair, and an arc from (s_i, t_i) to (s_j, t_j) when $s_j = t_i$. If this auxiliary graph contains a directed cycle, this implies that the original instance must be oriented in a way that creates a directed cycle, but since we assumed that G is mixed acyclic, this is impossible. Hence, the interesting case is when the auxiliary graph is a DAG, therefore there exists a demand pair (s_i, t_i) such that for all other demands (s_j, t_j) we have, $t_j \neq s_i$

We can apply the following method: take such a pair (s_i, t_i) (as described in the previous paragraph). If $s_i \in V \setminus S$, then orient all undirected edges between s_i and S from s_i to S. If the demand (s_i, t_i) is now satisfied, that is, there is a path from s_i to t_i using arcs and oriented edges, we remove this demand.

If (s_i, t_i) is still unsatisfied, if $t_i \in V \setminus S$, then orient all undirected edges between t_i and S from S to t_i.

We have now handled the demand (s_i, t_i): if it is satisfied after the steps above, we remove it from the instance and continue with the remaining demands; otherwise this demand cannot be satisfied and we reject. We eat this method until there are no unsatisfied terminals.

This method is correct: indeed, from Claim 12, after the guessing steps, the satisfaction of (s_i, t_i) only depends on the orientation of the undirected edges incident on $\{s_i, t_i\} \setminus S$. Furthermore, from the same claim, if $s_i \in V \setminus S$, in any feasible solution, its undirected edges will only be used to satisfy pairs of terminals that contain s_i. But our choice of terminals

ensures that for any such pair (s_j, t_j), $s_i \neq t_j$, so $s_i = s_j$. So in a feasible solution respecting the guessing steps, if an undirected edge of s_i is used to satisfy such a pair, it will be oriented from s_i to S, so it was safe to select this orientation. For the second step, if (s_i, t_i) is still not satisfied, then from Claim 12, the only way to satisfy it while respecting the guessing steps is, if $t_i \in V \setminus S$, to direct one undirected edge between t_i and S from S to t_i. But from Claim 11, they all have the same orientation, so it was safe to orient all of them from S to t_i. If (s_i, t_i) is still unsatisfied after these steps, then, from Claim 12, there is no way to satisfy those terminals while respecting the guessing steps.

The above method can be applied in time $n^{\mathcal{O}(1)}$, and must be applied at most n^2 times. In the end, we have an algorithm of complexity $n^{\mathcal{O}(\mathrm{vc}^2)}$. ◀

6 Parameterized by Number of Edges or Arcs

Arguably one of the most natural ways to structurally parameterize Steiner Orientation is to consider either the number of undirected edges $|E|$ or the number of arcs $|A|$ as a parameter. Our goal in this section is to investigate the complexities of these two parameterizations.

We begin with the parameterization by $|E|$, which is trivially FPT: one can simply guess the orientation of each edge and then check if this gives a valid solution. The complexity is therefore $2^{|E|}n^{\mathcal{O}(1)}$. Our contribution is to observe that this trivial algorithm is perhaps optimal, as if one could obtain an algorithm of complexity $(2-\varepsilon)^{|E|}n^{\mathcal{O}(1)}$, then the Strong Exponential Time Hypothesis (SETH) would be false. Recall that, informally, the SETH states that there is no algorithm for SAT running faster than the 2^n brute-force algorithm enumerating all assignments. We show the lower bound by reusing a simple reduction from SAT which appeared in [1].

▶ **Theorem 13.** *If there exists an $\varepsilon > 0$ such that* Steiner Orientation *can be solved in time $(2-\varepsilon)^{|E|}n^{\mathcal{O}(1)}$, then the satisfiability of a CNF formula ϕ with n variables can be decided in time $(2-\varepsilon)^n|\phi|^{\mathcal{O}(1)}$, hence the SETH is false.*

The main result of this section is then to establish that the parameterization by $|A|$ also gives a fixed-parameter tractable problem, and indeed that we are again able to obtain a single-exponential parameter dependence. This parameterization is significantly more challenging, so we will need to rely on structural properties obtained by any graph where $|A|$ is small after we apply some simple reduction rules. The main result we obtain is stated below.

▶ **Theorem 14.** Steiner Orientation *admits a $2^{\mathcal{O}(|A|)}n^{\mathcal{O}(1)}$ algorithm.*

Our high-level strategy will be made up of two parts. First, we define a (very) restricted special case of Steiner Orientation, where, notably, all undirected components are paths (and we have some additional restrictions). With those severe restrictions, this case becomes simpler. In particular, we will show that it admits an polynomial algorithm, via a reduction to 2-SAT. Second, we will show how a branching procedure can reduce any initial instance into a collection of at most $2^{6|A|} = 64^{|A|}$ instances of the restricted problem. The result will then follow by composing the two algorithms. In order to ease presentation, we start with the algorithm for the restricted case in Section 6.1 and then give the algorithm for parameter $|A|$ in Section 6.2.

6.1 A Restricted Case

We define a restricted case of Steiner Orientation as follows:

▶ **Definition 15.** *An instance $G = (V, E, A)$ of* Steiner Orientation *is called* restricted *if we have the following: all vertices $v \in V$ of mixed degree at least 3 are not incident on any unoriented edges, only arcs.*

Intuitively, a restricted instance which is also mixed acyclic is an instance where each non-trivial undirected component is a path (because the maximum degree induced by E is 2 and we have no cycles), the internal vertices of each such path have no arcs connecting them to the outside, and furthermore each endpoint of such path has only one arc. In the following, we will show that these restricted instances are polynomial time solvable.

Our strategy to handle restricted instances of Steiner Orientation will be the following: first, we simplify our instance by making several observations allowing us to orient several edges deterministically. Then, we encode the obtained instance as a 2-SAT formula, by defining for each terminal in an undirected path a variable which encodes whether this vertex can reach (or be reached by) one endpoint of its path or the other in the final solution, and verifying that each terminal pair is satisfied.

▶ **Lemma 16.** *There is an algorithm that takes a restricted* Steiner Orientation *instance $G = (V, E, A)$ that is mixed acyclic and decides if a solution exists in time $n^{\mathcal{O}(1)}$.*

6.2 FPT algorithm by Number of Arcs

Proof of Theorem 14. We begin by applying Proposition 1 and Proposition 2 exhaustively. This does not increase $|A|$, so we can assume that the input is mixed acyclic and has no vertices of degree at most 1. Our algorithm is now the following: take all vertices of mixed degree 3 or more, and for each possible orientation of their edges, create a new instance. We claim that we obtain at most $2^{6|A|}$ restricted instances, such that at least one of them will have a solution if and only if the original instance did.

First, we argue that if we perform this branching on all vertices of mixed degree 3 or more, we obtain an instance conforming to Definition 15. Indeed, after the orientation, all vertices of mixed degree 3 are not incident on any unoriented edges, only arcs.

In order to bound the number of instances, let us count the number of edges to orient. We will need the following lemma:

▷ Claim 17. Let $T = (V, E)$ be an undirected forest, $V_3 \subseteq V$ be the set of vertices of degree at least 3 in T and $V_1 \subseteq V$ the set of vertices of degree 1. Then, $\sum_{v \in V_3} d(v) \leq 3|V_1|$.

Armed with this claim, let us continue: take $T \subseteq G[E]$ composed of the non-trivial connected components of $G[E]$. As G is mixed acyclic, it is a forest. Furthermore, as G does not contain any leaf, each leaf of T has at least one arc. Let V_0 be the set of leaves of T with a unique arc, V_1 be the set of leaves of T with two arcs or more, V_2 the set of vertices with two edges and at least one arc, and V_3 the set of vertices with at least 3 edges. We need to orient the unoriented edges of V_1, V_2 and V_3, which make $|V_1| + 2|V_2| + \sum_{v \in V_3} d(v)$ edges. But, as each arc has only two endpoints, $|V_0| + 2|V_1| + |V_2| \leq 2|A|$. And, from the previous claim, $\sum_{v \in V_3} d(v) \leq 3(|V_0| + |V_1|)$. Therefore, $|V_1| + 2|V_2| + \sum_{v \in V_3} d(v) \leq 3|V_0| + 4|V_1| + 2|V_2| \leq 3(|V_0| + 2|V_1| + |V_2|) \leq 6|A|$:, so we need to orient at most $6|A|$ edges, thus obtaining at most $2^{6|A|}$ restricted instances. ◀

7 Kernelization

In this section we show that any instance $(G, \mathcal{T})$ of Steiner Orientation admits a polynomial kernel with respect to parameter $\mathrm{vc} + k$, where vc is the size of a minimum vertex cover of G and k is the number of terminal pairs.

In particular we will prove that:

▶ **Theorem 18.** Steiner Orientation *admits a kernel of order* $\mathcal{O}(\text{vc}^2 + k)$*, where* vc *is the size of the minimum vertex cover of* G *and* k *is the number of terminal pairs.*

The high-level idea is that in the final kernel we will keep (i) the vertices of the vertex cover (which are $\mathcal{O}(\text{vc})$) (ii) all terminals (which are $\mathcal{O}(k)$) (iii) $\mathcal{O}(\text{vc}^2)$ additional vertices from the independent set. Similarly to the XP algorithm for parameter vertex cover, we use the idea that once we have fixed all paths of length at most 2 between vertices of the vertex cover, this is sufficient to fully determine connectivity in the graph. Therefore, at most $\mathcal{O}(\text{vc}^2)$ vertices from the independent set are needed to form the solution. The challenge is then to identify a set of that size which is always sufficient and we achieve this via a reduction to Maximum Matching.

8 FPT by Treewidth plus Number of Terminals

Our goal in this section is to show that even though Steiner Orientation is NP-hard for graphs of constant treewidth and also W[1]-complete parameterized by the number of terminals k, it is in fact FPT when parameterized by the two parameters together. In order to ease presentation we will avoid giving an argument based on a dynamic programming algorithm and will prefer to instead rely on Courcelle's theorem [5]. This will of course have the disadvantage that we obtain an algorithm with running time of the form $f(\text{tw}, k)n^{\mathcal{O}(1)}$, where the function f is not explicitly bounded (and could be a tower of exponentials). This is sufficient for our purpose of classifying the complexity of this case as FPT, but we leave it as an interesting open problem to determine the best parameter dependence f one can achieve. We in fact establish the following slightly stronger claim:

▶ **Theorem 19.** Steiner Orientation *is FPT parameterized by the treewidth of the augmented underlying graph, that is, the treewidth of the graph obtained by taking the underlying graph of the input and adding for each terminal pair* (s_i, t_i) *the edge* $s_i t_i$ *to the graph.*

We observe that Theorem 19 implies that Steiner Orientation is FPT parameterized by treewidth plus k (the number of terminals), because adding the extra edges to the underlying graph can increase its treewidth by at most k.

9 Conclusion

Even though we have characterized the complexity of Steiner Orientation for all possible values of treewidth (the only polynomial case is when the input is a forest), our NP-completeness only applies for graphs of pathwidth 3: is the problem in P or NP-complete for graphs of pathwidth 2. A similar question can be asked for feedback vertex set, where we show hardness when the feedback vertex set has size at least 2: does the problem become tractable on graphs which are one vertex away from being forests? On the parameterized complexity side, even though we obtain a $2^{\mathcal{O}(|A|)}n^{\mathcal{O}(1)}$ algorithm, the base in the exponential is large (64), so improving this is a natural question.

References

1 Esther M. Arkin and Refael Hassin. A note on orientations of mixed graphs. *Discret. Appl. Math.*, 116(3):271–278, 2002. doi:10.1016/S0166-218X(01)00228-1.

2 Hans L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theor. Comput. Sci.*, 209(1-2):1–45, 1998. doi:10.1016/S0304-3975(97)00228-4.

3 Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph classes: a survey*. SIAM, 1999.

4 Rajesh Chitnis, Andreas Emil Feldmann, and Ondrej Suchý. A tight lower bound for planar steiner orientation. *Algorithmica*, 81(8):3200–3216, 2019. doi:10.1007/S00453-019-00580-X.

5 Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

6 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.

7 Marek Cygan, Guy Kortsarz, and Zeev Nutov. Steiner forest orientation problems. *SIAM J. Discret. Math.*, 27(3):1503–1513, 2013. doi:10.1137/120883931.

8 Andreas Darmann and Janosch Döcker. On simplified NP-complete variants of Monotone 3-Sat. *Discret. Appl. Math.*, 292:45–58, 2021. doi:10.1016/J.DAM.2020.12.010.

9 Britta Dorn, Falk Hüffner, Dominikus Krüger, Rolf Niedermeier, and Johannes Uhlmann. Exploiting bounded signal flow for graph orientation based on cause-effect pairs. *Algorithms Mol. Biol.*, 6:21, 2011. doi:10.1186/1748-7188-6-21.

10 Michael Elberfeld, Danny Segev, Colin R. Davidson, Dana Silverbush, and Roded Sharan. Approximation algorithms for orienting mixed graphs. *Theor. Comput. Sci.*, 483:96–103, 2013. doi:10.1016/J.TCS.2012.03.044.

11 Iftah Gamzu and Moti Medina. Improved approximation for orienting mixed graphs. *Algorithmica*, 74(1):49–64, 2016. doi:10.1007/S00453-014-9932-2.

12 E. Mark Gold. Complexity of automaton identification from given data. *Inf. Control.*, 37(3):302–320, 1978. doi:10.1016/S0019-9958(78)90562-4.

13 David G. Harris and N. S. Narayanaswamy. A faster algorithm for vertex cover parameterized by solution size. In Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov, editors, *41st International Symposium on Theoretical Aspects of Computer Science, STACS 2024, March 12-14, 2024, Clermont-Ferrand, France*, volume 289 of *LIPIcs*, pages 40:1–40:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.STACS.2024.40.

14 Rafael Hassin and Nimrod Megiddo. On orientations and shortest paths. *Linear Algebra and its Applications*, 114-115:589–602, 1989.

15 Florian Hörsch. Maximum reachability orientation of mixed graphs, 2025. doi:10.48550/arXiv.2506.16171.

16 Alexander Medvedovsky, Vineet Bafna, Uri Zwick, and Roded Sharan. An algorithm for orienting graphs based on cause-effect pairs and its applications to orienting protein networks. In *Algorithms in Bioinformatics, 8th International Workshop, WABI 2008. Proceedings*, volume 5251 of *Lecture Notes in Computer Science*, pages 222–232. Springer, 2008. doi:10.1007/978-3-540-87361-7_19.

17 Marcin Pilipczuk and Magnus Wahlström. Directed multicut is W[1]-hard, even for four terminal pairs. *ACM Trans. Comput. Theory*, 10(3):13:1–13:18, 2018. doi:10.1145/3201775.

18 Mehdy Roayaei. On the parameterized complexity of the problem of inferring protein-protein interaction directions based on cause-effect pairs. *Netw. Model. Anal. Health Informatics Bioinform.*, 8(1):11, 2019. doi:10.1007/S13721-019-0194-4.

19 Dana Silverbush, Michael Elberfeld, and Roded Sharan. Optimally orienting physical networks. *J. Comput. Biol.*, 18(11):1437–1448, 2011. doi:10.1089/CMB.2011.0163.

20 Michał Włodarczyk. Parameterized inapproximability for steiner orientation by gap amplification. In *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020*, volume 168 of *LIPIcs*, pages 104:1–104:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.ICALP.2020.104.

Reachability of Independent Sets and Vertex Covers Under Extended Reconfiguration Rules

Shuichi Hirahara
National Institute of Informatics, Tokyo, Japan

Naoto Ohsaka
CyberAgent, Inc., Tokyo, Japan

Tatsuhiro Suga
Graduate School of Information Sciences, Tohoku University, Sendai, Japan

Akira Suzuki
Center for Data-Driven Science and Artificial Intelligence, Tohoku University, Sendai, Japan

Yuma Tamura
Graduate School of Information Sciences, Tohoku University, Sendai, Japan

Xiao Zhou
Graduate School of Information Sciences, Tohoku University, Sendai, Japan

Abstract

In reconfiguration problems, we are given two feasible solutions to a graph problem and asked whether one can be transformed into the other via a sequence of feasible intermediate solutions under a given reconfiguration rule. While earlier work focused on modifying a single element at a time, recent studies have started examining how different rules impact computational complexity.

Motivated by recent progress, we study Independent Set Reconfiguration (ISR) and Vertex Cover Reconfiguration (VCR) under the k-Token Jumping (k-TJ) and k-Token Sliding (k-TS) models. In k-TJ, up to k vertices may be replaced, while k-TS additionally requires a perfect matching between removed and added vertices. It is known that the complexity of ISR crucially depends on k, ranging from PSPACE-complete and NP-complete to polynomial-time solvable.

In this paper, we further explore the gradient of computational complexity of the problems. We first show that ISR under k-TJ with $k = |I| - \mu$ remains NP-hard when μ is any fixed positive integer and the input graph is restricted to graphs of maximum degree 3 or planar graphs of maximum degree 4, where $|I|$ is the size of feasible solutions. In addition, we prove that the problem belongs to NP not only for $\mu = O(1)$ but also for $\mu = O(\log |I|)$. In contrast, we show that VCR under k-TJ is in XP when parameterized by $\mu = |S| - k$, where $|S|$ is the size of feasible solutions. Furthermore, we establish the PSPACE-completeness of ISR and VCR under both k-TJ and k-TS on several graph classes, for fixed k as well as superconstant k relative to the size of feasible solutions.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Problems, reductions and completeness

Keywords and phrases combinatorial reconfiguration, extended reconfiguration rule, independent set reconfiguration, vertex cover reconfiguration, PSPACE-completeness, NP-completeness

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.39

Related Version *Full Version*: https://arxiv.org/abs/2510.24226

Funding This work was partially supported by Institute of Mathematics for Industry, Joint Usage/Research Center in Kyushu University. (FY2024 Workshop (II) "Theory of Combinatorial Reconfiguration and Beyond" (2024a037))
Akira Suzuki: Partially supported by JSPS KAKENHI Grant Numbers JP25K14980.
Yuma Tamura: Partially supported by JSPS KAKENHI Grant Number JP25K21148.

Acknowledgements We are grateful to the anonymous referees for their helpful comments.

© Shuichi Hirahara, Naoto Ohsaka, Tatsuhiro Suga, Akira Suzuki, Yuma Tamura, and Xiao Zhou;
licensed under Creative Commons License CC-BY 4.0

1 Introduction

Reconfiguration problems ask whether it is possible to reach a target state from an initial state by gradually transforming one feasible solution into another, where each intermediate solution must also be feasible. At each step, the current solution can be changed to an "adjacent" one, as defined by a given restriction known as a *reconfiguration rule*. One of the most well-known examples is the *15-puzzle*, where the rule permits sliding a tile into an adjacent empty space. Reconfiguration problems have been extensively studied in the context of classical graph problems involving feasible solutions such as independent sets [3, 4, 6, 7, 11, 18, 20, 23, 27], cliques [22], vertex covers [29], dominating sets [16, 36], and so on (see surveys [31, 38]). Three standard reconfiguration models have been widely studied in the literature: *token jumping* (TJ) [20, 23], *token sliding* (TS) [3, 6, 18, 20], and *token addition/removal* (TAR) [20, 23] models. In the TJ model, one can simultaneously remove any vertex from the current solution and add any vertex outside it. The TS model is a restricted version of TJ, where the removed vertex and the added vertex must be adjacent. In the TAR model, vertices can be added or removed as long as the resulting set remains above (or below) a specified size threshold.

Reconfiguration problems on graphs under those rules have attracted attention in theoretical computer science, and the computational complexity of such problems has been settled [20, 23, 30, 40]. Besides, the field of reconfiguration problems is in the course of trying to apply the theoretical viewpoint to functional implementation for practical use [9, 21, 37, 41].

However, one may find that, in practical scenarios, conventional rules such as changing one element at a time may be too restrictive. For example, reconfiguring a monitoring system or an infrastructure network often requires multiple simultaneous changes due to physical or operational constraints. Even if the system cannot be reconfigured under standard one-element rules, in practice, it is unjustified to conclude that it is "unreachable." More flexible reconfiguration processes – such as those allowing multiple simultaneous changes – more accurately reflect the feasibility of real-world systems.

Motivated by recent progress and aiming to address the emerging issues, several studies have begun to analyze the computational complexity of reconfiguration problems under extended reconfiguration rules [10, 12, 17, 26, 35].

We study the reconfiguration problem under the extended rules, k-Token Jumping (k-TJ) and k-Token Sliding (k-TS) [10, 35], which allow the simultaneous exchange of up to k vertices.

1.1 Our Problems

In this paper, all graphs are simple and undirected. For two sets A and B, the set difference $A \setminus B$ is $\{x \in A \colon x \notin B\}$, and $A \triangle B$ denotes the symmetric difference between A and B, that is, $(A \setminus B) \cup (B \setminus A)$. For two vertex subsets A and B of a graph G with $|A| = |B|$, we say that they are *adjacent* under *k-Token Jumping* (k-TJ) if $|A \triangle B| \leq 2k$. On the other hand, they are said to be *adjacent* under *k-Token Sliding* (k-TS) if $|A \triangle B| \leq 2k$ and there exists a perfect matching between $A \setminus B$ and $B \setminus A$ in the graph G. Intuitively, the transformation from A to B can be seen as moving tokens placed on the vertices in the symmetric difference $A \triangle B$. Under k-TJ, up to k tokens can be moved simultaneously to any vertices in G. In contrast, under k-TS, up to k tokens can be moved simultaneously along the edges of G. Note that 1-TJ and 1-TS coincide with TJ and TS, respectively.

Independent Set and Vertex Cover are NP-complete problems [15] that are thoroughly explored in graph theory. We define their reconfiguration variants, that is, Independent Set Reconfiguration (ISR) and Vertex Cover Reconfiguration (VCR).

Recall that a set of vertices in a graph G is called an *independent set* if no two vertices in the set are adjacent in G, and a *vertex cover* if every edge in G has at least one endpoint in the set.

In the Independent Set Reconfiguration problem, we are given a graph G, two independent sets I and J of G (representing the "initial" and "target" configurations of tokens, respectively) such that $|I| = |J|$, and a reconfiguration rule $\mathsf{R} \in \{k\text{-}\mathsf{TJ}, k\text{-}\mathsf{TS}\}$. Then, the problem asks whether there exists a sequence $\sigma = \langle I = I_0, I_1, \ldots, I_\ell = J \rangle$ of independent sets of G, where any two consecutive independent sets in σ are adjacent under R. Similarly, in Vertex Cover Reconfiguration, we are given a graph G, two vertex covers S and T of G (representing the "initial" and "target" configurations of tokens, respectively) such that $|S| = |T|$, and a reconfiguration rule $\mathsf{R} \in \{k\text{-}\mathsf{TJ}, k\text{-}\mathsf{TS}\}$. Then, the problem asks whether there exists a sequence $\sigma = \langle S = S_0, S_1, \ldots, S_\ell = T \rangle$ of vertex covers of G, where any two consecutive vertex covers in σ are adjacent under R. In each problem, we refer to such a sequence of independent sets or vertex covers as a *reconfiguration sequence*, where ℓ is the length of the reconfiguration sequence. Furthermore, we say that two vertex sets are *reconfigurable* under R if there exists a reconfiguration sequence between them under R. Formally, the two problems are defined as follows.

Problem Independent Set Reconfiguration (ISR)
Input A simple undirected graph G, two independent sets I and J of G with the same size, and a reconfiguration rule $\mathsf{R} \in \{k\text{-}\mathsf{TJ}, k\text{-}\mathsf{TS}\}$.
Output Are I and J reconfigurable under R?

Problem Vertex Cover Reconfiguration (VCR)
Input A simple undirected graph G, two vertex covers S and T of G with the same size, and a reconfiguration rule $\mathsf{R} \in \{k\text{-}\mathsf{TJ}, k\text{-}\mathsf{TS}\}$.
Output Are S and T reconfigurable under R?

Related work. ISR under TJ and TS is PSPACE-complete even for planar graphs of maximum degree 3 and bounded bandwidth [18, 39, 40], and perfect graphs [23]. Under TJ, it is known that any two independent sets of an even-hole-free graph are reconfigurable [23]. Under TS, the problem remains PSPACE-complete for split graphs [3], while it can be solved in polynomial time for interval graphs [4]. For claw-free graphs, ISR under both TJ and TS can be solved in polynomial time [6]. For bipartite graphs, interestingly, ISR is NP-complete under TJ, while PSPACE-complete under TS [27]. Note that ISR and VCR under TJ and TS are essentially equivalent due to their complementary relationship; therefore, their computational complexities coincide.

In [35], several results are presented for ISR under the k-TJ and k-TS rules. The paper shows that ISR under both k-TJ and k-TS is PSPACE-complete on perfect graphs for any fixed integer $k \geq 2$. Furthermore, k-TS and TS are essentially equivalent on even-hole-free graphs [35]. As a result, the computational complexity of ISR under k-TS on several graph classes contained within the class of even-hole-free graphs follows from the results under TS.

Křišťan et al. studied several reconfiguration problems, including ISR and VCR, under the (k, d)-*Token Jumping* model [26]. In this model, k tokens can move simultaneously, and each token can travel within a distance of d. The (k, d)-Token Jumping model may appear to generalize the reconfiguration rules k-TJ and k-TS; however, the settings are slightly different. The (k, d)-Token Jumping model is defined by a bijection between the current configuration and the next configuration. Consequently, a token can move to a vertex currently occupied by another token, as long as the latter token moves to a different vertex in the same step. In

contrast, the definitions of k-TJ and k-TS are based on the symmetric difference between configurations. In these models, a token is prohibited from moving to a vertex that is currently occupied by another token. Although both the (k, d)-Token Jumping model and k-TJ (resp. k-TS) are natural generalizations of TJ (resp. TS), the difference between them significantly impacts the computational complexity of problems. In fact, for the number t of tokens, VCR under the $(t, 1)$-Token Jumping model can be solved in polynomial time [26], while VCR under the k-TS is PSPACE-complete when $k = t$ [35].

1.2 Our Contribution

We research the computational complexity of ISR and VCR under k-TJ and k-TS. An overview of our results is provided in Tables 1 and 2. The numbering of theorems follows the system starting from Section 3. Therefore, in this section, note that the numbering does not begin consecutively.

The results when parameterized by the guaranteed value

We first investigate the complexity of ISR under k-TJ when k is defined as $|I| - \mu$, where I is the initial independent set of an ISR instance and μ is a parameter referred to as the *guaranteed value* [28]. This parameter measures how far the instance is from the trivial case: when $\mu = 0$ (that is, $k = |I|$), ISR under k-TJ becomes trivial, as all vertices in the independent set can be replaced simultaneously. Hence, we are interested in the computational complexity of the problem when μ is small. It was shown in [35] that even for any fixed positive integer μ, ISR under k-TJ is NP-complete when $k = |I| - \mu$. However, the reduction presented in [35] introduces a large number of edges. From a practical standpoint, it is particularly important to understand the computational complexity of the problem on sparse graphs, such as planar graphs or graphs with bounded degree.

To answer this question, we present NP-hardness results for these sparse classes.

▶ **Theorem 2.** *Let μ be any fixed positive integer.* ISR *under k-TJ is* NP*-hard for graphs G of maximum degree* 3 *when $k = |I| - \mu \geq 1$, where I is an initial independent set of G.*

▶ **Theorem 3.** *Let μ be any fixed positive integer.* ISR *under k-TJ is* NP*-hard for planar graphs G of maximum degree* 4 *when $k = |I| - \mu \geq 1$, where I is an initial independent set of G.*

Here, let us explain the main obstacle in the proofs of these theorems. Consider the case where $\mu = 1$. If the initial and target independent sets I and J satisfy $|I \cap J| \geq 1$, then the reconfiguration is trivial. Hence, to ensure that the instance is non-trivial, it is essential to construct I and J so that $I \cap J = \emptyset$. A common approach used in existing research is to employ a complete bipartite graph; however, this prevents the resulting graph from being sparse. A more delicate reduction is required to preserve the sparsity of the graph. As a key step toward this goal, we introduce a new variant of Exactly 3-SAT, which we call Internal Exactly 3-SAT, and show that it is NP-complete. In this variant, the input is a 3-CNF formula that is promised to be satisfiable under both the all-true and all-false assignments. The goal is to determine whether there exists a *mixed* satisfying assignment, that is, a satisfying assignment that is neither all-true nor all-false. We convert the all-true and all-false assignments to the initial and target independent sets, respectively. The existence of a mixed satisfying assignment corresponds to the reconfigurability from I to J via an independent set I' such that $|I' \cap I| \geq 1$ and $|I' \cap J| \geq 1$.

Table 1 The complexity of ISR under k-TJ for various graph classes and values of k. Here, I denotes an initial independent set, and Δ, bw, and cw denote the maximum degree, bandwidth, and clique-width, respectively, of a given n-vertex graph. Let $\varepsilon_0 \in (0, 1)$ be a fixed constant.

	k-TJ					
				$k = \lvert I\rvert - \mu$		
	$k = 1$ (TJ)	any const. $k \geq 2$	any $k \leq \varepsilon_0 \lvert I\rvert$	$\mu = O(\log \lvert I\rvert)$	any const. $\mu > 0$	$k = \lvert I\rvert$
general	PSPACE-c.		PSPACE-c. [Theorem 9]	open (in NP?)	NP-c. [35]	trivially always yes
bounded Δ	PSPACE-c. [18, 39, 40]	PSPACE-c. [Theorem 7]		NP-c. [Theorem 2]		
$\Delta = 3$						
planar $\cap$ $\Delta = o(\frac{n}{\log n})$			NP-h. [Theorem 3]			
planar $\cap$ $\Delta = 4$				[Theorem 4]		
planar $\cap$ $\Delta = 3$			open	open (NP-hard?)		
planar $\cap$ $\Delta = 3$ $\cap$ bounded bw				XP parameterized by $\mu = \lvert I\rvert - k$ [35]		
bounded cw						
perfect	PSPACE-c. [23]	PSPACE-c. [35]				
bipartite	NP-c. [27]	open				
claw-free	P [6, 23]	PSPACE-c. ($k = 2$) [Theorem 8]				
line						
even-hole-free	always yes [23]					

We next demonstrate that ISR under k-TJ with $k = |I| - \mu$ belongs to NP even when $\mu = O(\log |I|)$ for some graph classes. Since many reconfiguration problems are PSPACE-complete due to the potentially super-polynomial length of reconfiguration sequences, showing that a reconfiguration problem belongs to NP is non-trivial. It is known that ISR under k-TJ with $k = |I| - \mu$ is in NP when μ is constant [35]. We strengthen this result by proving NP-membership for specific graph classes under the condition $\mu = O(\log |I|)$.

▶ **Theorem 4.** *Let G be an input graph with n vertices, chromatic number $O(1)$ and maximum degree $o(\frac{n}{\log n})$, and let I be an initial independent set of G.* ISR *under k-TJ is in* NP *when $k = |I| - \mu \geq 1$ with any non-negative integer μ at most $O(\log |I|)$.*

In the proof of Theorem 4, we utilize the concept of an *intersecting family* of a set [13], which has been extensively studied in the field of extremal set theory. Based on this concept, we derive an upper bound on the length of the reconfiguration sequence.

Here, the following Theorem 6 constitutes another main result of our work.

▶ **Theorem 6.** VCR *under k-TJ is in* XP *for general graphs G when parameterized by $\mu = |S| - k \geq 0$, where S is an initial vertex cover of G.*

Recall that in any graph G, a vertex cover and an independent set are complementary: the complement of a vertex cover of G is an independent set of G. Thus, ISR and VCR are generally considered equivalent problems. However, our result for VCR in Theorem 6 stands in sharp contrast to the known results for ISR (see also Table 2). In the proof of Theorem 6, we design an XP algorithm based on a *clique-compressed reconfiguration graph* [35].

The results when k is constant

We establish the PSPACE-completeness of ISR for fixed k on several graph classes.

▶ **Theorem 7.** *Let $k \geq 2$ be any fixed positive integer.* ISR *under $\mathsf{R} \in \{k\text{-TS}, k\text{-TJ}\}$ is* PSPACE*-complete for planar graphs of maximum degree 3 and bounded bandwidth.*

Table 2 Comparison between the complexity of ISR and VCR under k-TJ on general graphs and graphs of maximum degree 3. The vertex subset A of the input graph represents the initial solution for both ISR and VCR. Specifically, A is the input independent set for ISR, and the input vertex cover for VCR. Let $\varepsilon_0 \in (0, 1)$ be some constant.

problems	graph classes	any $k \leq \varepsilon_0 \lvert A \rvert$	$k = \lvert A \rvert - \mu$	
			$\mu = O(\log \lvert A \rvert)$	any fixed $\mu \geq 1$
ISR	general	PSPACE-c.	open (in NP?)	NP-c. [35]
	maximum degree 3	[Theorem 9]	NP-c. [Theorem 2, Theorem 4]	
VCR	general	PSPACE-c.	XP [Theorem 6]	
	maximum degree 3	[Theorem 12]	parameterized by $\mu = \lvert A \rvert - k$	

Together with known results for the case $k = 1$ [18, 39, 40], our findings provide a complete characterization of the PSPACE-completeness of ISR under k-TJ and k-TS for every fixed positive integer k and planar graphs of maximum degree 3 and bounded bandwidth.

We further demonstrate the following Theorem 8.

▶ **Theorem 8.** ISR *under* $\mathsf{R} \in \{2\text{-TJ}, 2\text{-TS}\}$ *is* PSPACE*-complete for line graphs.*

This result stands in sharp contrast to the polynomial-time solvability of ISR under TJ and TS on claw-free graphs [6], which include line graphs as a special case.

The results when k is superconstant

We investigate the complexity of ISR under k-TJ when k is superconstant in the initial independent set size $|I|$. To this end, we construct a polynomial-time reduction from the "optimization variant" called MaxminISR to ISR. By using the PSPACE-hardness of approximating MaxminISR [19, 24, 32], we prove that ISR under k-TJ is PSPACE-complete on graphs of maximum degree 3 for a wide range of values of k, including $k = O(1)$, $\Theta(\log |I|)$, $\Theta(|I|^{O(1)})$, and $\Theta(|I|)$.

In particular, we show the following Theorem 9.

▶ **Theorem 9.** *There exists some constant* $\varepsilon_0 \in (0, 1)$ *such that* ISR *under* k-TJ *on graphs of maximum degree* 3 *is* PSPACE*-complete for any* k *satisfying the following condition: there exists a constant* c *such that* $k \leq \varepsilon_0 |I|$ *holds whenever* $|I| \geq c$*, where* I *is the initial independent set of the input graph.*

We show that a similar result holds for VCR.

▶ **Theorem 12.** *There exists some constant* $\varepsilon_0 \in (0, 1)$ *such that* VCR *under* k-TJ *on graphs of maximum degree* 3 *is* PSPACE*-complete for any* k *satisfying the following condition: there exists a constant* c *such that* $k \leq \varepsilon_0 |S|$ *holds whenever* $|S| \geq c$*, where* S *is the initial vertex cover of the input graph.*

Outline. The remainder of this paper is organized as follows. We begin with preliminaries in Section 2. In Section 3, we study the problems ISR and VCR under k-TJ, focusing on the guaranteed value μ. In Section 4, we demonstrate the PSPACE-completeness of these problems when k is fixed. Finally, in Section 5, we analyze the case where k is superconstant. Due to the space limitation, proofs marked $*$ are omitted and can be found in the full version of this paper.

2 Preliminaries

For a positive integer i, we write $[i] = \{1, 2, \ldots, i\}$. Let $G = (V, E)$ be a finite, simple, and undirected graph with the vertex set V and the edge set E. We use $V(G)$ and $E(G)$ to denote the vertex set and the edge set of G, respectively. For a vertex v of G, $N_G(v)$ and $N_G[v]$ denote the *open neighborhood* and the *closed neighborhood* of v, respectively, that is, $N_G(v) = \{u \in V(G) \colon uv \in E(G)\}$ and $N_G[v] = N_G(v) \cup \{v\}$. For a vertex set $X \subseteq V(G)$, we define $N_G(X) = \{v \in V(G) \setminus X \colon uv \in E(G), u \in X\}$ and $N_G[X] = N_G(X) \cup X$. A subgraph of G is a graph G' such that $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$. For a subset $S \subseteq V(G)$, $G[S]$ denotes the subgraph induced by S. The *degree* of a vertex v in a graph G is the number of edges incident to v. The *maximum degree* of G is the largest degree among all vertices in G. A sequence $\langle v_0, v_1, \ldots, v_\ell \rangle_G$ of vertices of a graph G, where $v_{i-1}v_i \in E$ for each $i \in [\ell]$, is called a *path* if all the vertices are distinct. It is called a *cycle* if $v_0, v_1, \ldots, v_{\ell-1}$ are distinct and $v_0 = v_\ell$. The value ℓ is referred to as the *length* of the path or cycle. A graph G is said to be *connected* if there exists a path between u and v for any pair of vertices $u, v \in V(G)$. The *chromatic number* of G is the smallest positive integer c such that G has a *c-coloring*, where a c-coloring of a graph G is a mapping $f \colon V(G) \to [c]$ such that $f(u) \neq f(v)$ if $uv \in E(G)$. The *bandwidth* of a graph $G = (V, E)$ is the minimum integer b such that there exists a bijection $f \colon V \to \{1, \ldots, |V|\}$ satisfying $|f(u) - f(v)| \leq b$ for every edge $uv \in E(G)$.

We conclude this section with the following simple remark. We can observe that the problems ISR and VCR can be solved in nondeterministic polynomial space. Thus, by applying Savitch's Theorem [34], these problems are in PSPACE. Consequently, to establish PSPACE-completeness of ISR and VCR, it suffices to prove PSPACE-hardness.

3 When Parameterized by a Guaranteed Value

In this section, we discuss ISR and VCR under k-TJ, focusing on the guaranteed value.

3.1 ISR

We investigate the NP-completeness of ISR under k-TJ when $k = |I| - \mu$, where I is the initial independent set of an ISR instance, and μ is a parameter called the *guaranteed value*. Hereafter, we use I to denote the initial independent set of an ISR instance. For any fixed positive integer μ, it is known that ISR under k-TJ is NP-complete when $k = |I| - \mu$ [35]. In Section 3.1.1, we show that ISR under k-TJ remains NP-hard even when the input graph is restricted to graphs of maximum degree 3, or to planar graphs of maximum degree 4, where $k = |I| - \mu$ for any fixed positive integer μ. Then, in Section 3.1.2, we prove that the problem remains in NP even when $\mu = O(\log |I|)$ and an input graph is a graph of bounded chromatic number and maximum degree $o(\frac{n}{\log n})$, where n is the number of vertices in the input graph.

3.1.1 NP-hardness

We show the NP-hardness of ISR under k-TJ for graphs of maximum degree 3 and planar graphs of maximum degree 4 when $k = |I| - \mu$ with any fixed positive integer μ. To this end, we construct a chain of reductions. As a source problem of our reduction, we will use Exactly 3-SAT (E3-SAT for short), which is NP-complete [15]. Firstly, E3-SAT is reduced to Internal Exactly 3-SAT (IntE3-SAT for short), a new variant of E3-SAT (to the best of our knowledge). Afterward, IntE3-SAT is reduced to our problem.

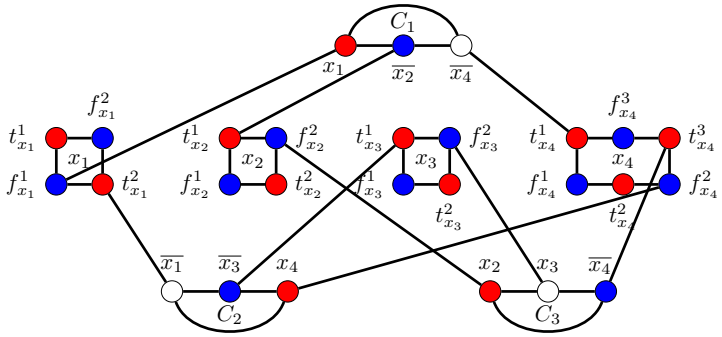


Figure 1 Illustration of the construction of G from a Boolean formula ϕ' used in the proof of Theorem 2, where $\phi' = (x_1 \vee \overline{x_2} \vee \overline{x_4}) \wedge (\overline{x_1} \vee \overline{x_3} \vee x_4) \wedge (x_2 \vee x_3 \vee \overline{x_4})$. The blue marked tokens are on I, and the red marked tokens are on J.

Here, let us define E3-SAT and IntE3-SAT. Let $\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ be a Boolean formula in conjunctive normal form (CNF), and $X = \{x_1, \ldots, x_n\}$ be the variable set of ϕ. Each clause C_i for $i \in [m]$ in ϕ is a disjunction of literals and each literal appears as either a positive form x_j or a negative form $\overline{x_j}$ for a variable $x_j \in X$. A variable assignment is a mapping $b\colon X \to \{T, F\}$. We say that a variable assignment *satisfies* ϕ if ϕ evaluates to T. A CNF formula ϕ is called an Ek-CNF formula if any clause in ϕ consists of exactly k literals. Given an E3-CNF formula ϕ, E3-SAT asks whether there exists a variable assignment that satisfies ϕ.

A variable assignment b is called an *all-T assignment* if $b(x) = T$ for all $x \in X$, and an *all-F assignment* if $b(x) = F$ for all $x \in X$. Otherwise, it is called *mixed*. An Ek-CNF formula ϕ is called *sandwiched* if all-T and all-F assignments satisfy ϕ. In other words, any clause of a sandwiched Ek-CNF formula ϕ contains both positive and negative literals. Given a sandwiched Ek-CNF formula ϕ, IntEk-SAT asks whether there exists a mixed variable assignment that satisfies ϕ. We first show that IntE3-SAT is NP-complete by reducing E3-SAT.

▶ **Lemma 1.** IntE3-SAT *is* NP*-complete.*

For the proof, refer to Section 3.1.1.1. By the polynomial-time reduction from IntE3-SAT, we prove the following Theorem 2.

▶ **Theorem 2.** *Let μ be any fixed positive integer.* ISR *under k-*TJ *is* NP*-hard for graphs G of maximum degree* 3 *when $k = |I| - \mu \geq 1$, where I is an initial independent set of G.*

Proof. We use a polynomial-time reduction from IntE3-SAT. Let ϕ' be an instance of IntE3-SAT and X' be the set of variables of ϕ'. We will construct an instance $(G, I, J, k\text{-TJ})$ of ISR under k-TJ where $k = |I| - 1$ (see Figure 1 for an illustration), and then modify for any fixed $\mu \geq 1$.

For each variable $x \in X'$, let a_x denote the number of clauses of ϕ' in which x appears as a literal. For each variable $x \in X'$, we set up a *variable gadget* Y_x, which is defined as a cycle with $2a_x$ vertices (note that if x appears only once, then Y_x is a path with two vertices). The vertices in Y_x are labeled with $t^1_x, f^1_x, t^2_x, f^2_x, \ldots, t^{a_x}_x, f^{a_x}_x$ in a counterclockwise order. Intuitively, t^i_x and f^i_x for each $i \in [a_x]$ correspond to T and F assignments for x, respectively. We refer to a vertex of Y_x corresponding to T (resp. F) assignment to x as a *true vertex* (resp. *false vertex*). For each clause C, we set up a *clause gadget* K_C, which is defined as

a complete graph with three vertices. The vertices in K_C correspond to the three literals in C. We refer to a vertex of K_C corresponding to a positive literal (resp. negative literal) of C as a *positive vertex* (resp. *negative vertex*). Finally, we connect variable gadgets and clause gadgets with edges as follows: For a vertex v in a clause gadget corresponding to a literal l of a variable x with occurrence $i \in [a_x]$, connect v to t_x^i if the literal is negative; otherwise, connect v to f_x^i. Let G be the obtained graph. Observe that G is a graph of maximum degree 3. Let I be a set of all false vertices in variable gadgets and a negative vertex arbitrarily chosen from each clause gadget. Let J be a set of all true vertices in variable gadgets and a positive vertex arbitrarily chosen from each clause gadget. Note that such I and J always exist as each clause has both positive and negative literals due to the definition of IntE3-SAT. Finally, we set $k = |I| - 1$.

If $\mu \geq 2$, then we also add $\mu - 1$ isolated vertices to G. Let V^* be the set of all added vertices. Then let G^* be an obtained graph, $I^* = I \cup V^*$, and $J^* = J \cup V^*$. We also set $k = |I^*| - \mu$. Note that I is a maximum independent set of G and thus any independent set with size $|I^*| = |I| + |V^*|$ of G^* contains V^*. This allows us to move at most $k = |I^*| - \mu = |I| + |V^*| - \mu = |I| - 1$ tokens simultaneously only on G. Therefore, $(G, I, J, (|I|-1)\text{-TJ})$ is a yes-instance if and only if $(G^*, I^*, J^*, (|I^*|-\mu)\text{-TJ})$ is a yes-instance.

For this reason, we discuss only the case when $\mu = 1$ here. The instance $(G, I, J, k\text{-TJ})$ of ISR under k-TJ is clearly obtained in polynomial time. To complete our reduction, we will show that ϕ' is a yes-instance of IntE3-SAT if and only if $(G, I, J, k\text{-TJ})$ is a yes-instance of ISR under k-TJ where $k = |I| - 1$.

Assume that there is a variable assignment b' that is mixed and satisfies ϕ'. We construct an independent set I' of G as follows. For a variable x_i with $i \in [n]$, if x_i is assigned T, then all true vertices of Y_{x_i} are contained in I'. Similarly, if x_i is assigned F, then all false vertices of Y_{x_i} are contained in I'. Since those vertices are chosen according to b', one vertex in each clause gadget that corresponds to a literal evaluating T can also be contained in I'. Furthermore, since b' is mixed, we have $|I \cap I'| \geq 1$ and $|I' \cap J| \geq 1$. Therefore, a reconfiguration sequence $\sigma = \langle I, I', J \rangle$ exists.

Conversely, assume that there is a reconfiguration sequence $\sigma = \langle I = I_0, I_1, \ldots, I_\ell = J \rangle$. Consider an integer $i \in [\ell]$ such that tokens on I_i are placed on true vertices for the first time. In other words, all tokens of I_{i-1} in variable gadgets are on false vertices. Since any positive vertex of any clause gadget is adjacent to a false vertex of a variable gadget, all positive vertices are not in I_{i-1}. Then, we claim that not all true vertices are in I_i. For the sake of contradiction, assume that all true vertices are in I_i. Then, since each negative vertex is adjacent to a true vertex, each token of I_i in clause gadgets is on a positive vertex. Thus, since I_{i-1} contains only false vertices and negative vertices, $I_{i-1} \cap I_i = \emptyset$. However, $|I_{i-1} \cap I_i| \geq 1$ must hold because they are adjacent under k-TJ, which is a contradiction. Therefore, we conclude that I_i contains both true and false vertices. From our construction of the variable gadgets, either true or false vertices are in I_i for each variable gadget. For each $j \in [n]$, let b' be a variable assignment such that x_j is assigned T if and only if true vertices of Y_{x_j} are in I_i. Then, literals of clauses in ϕ' corresponding to vertices in I_i evaluate T from our construction, that is, b' satisfies ϕ'. Therefore, ϕ' is a yes-instance of IntE3-SAT. ◀

By Theorem 2, we have proven the NP-hardness of ISR under k-TJ on graphs of maximum degree 3 unless we can move all tokens. We now proceed to the NP-hardness of ISR under k-TJ on planar graphs of maximum degree 4. The constructed graph G in the proof of Theorem 2 may have some edges crossing on a plane. We will eliminate these crossings by replacing them with a *crossover gadget* (as shown in Figure 2). A crossover gadget consists of eight vertices $u'_1, u'_2, v'_1, v'_2, w_1, \ldots, w_4$ and twelve edges: u'_1w_1, u'_1w_4, u'_2w_2, u'_2w_3, v'_1w_1,

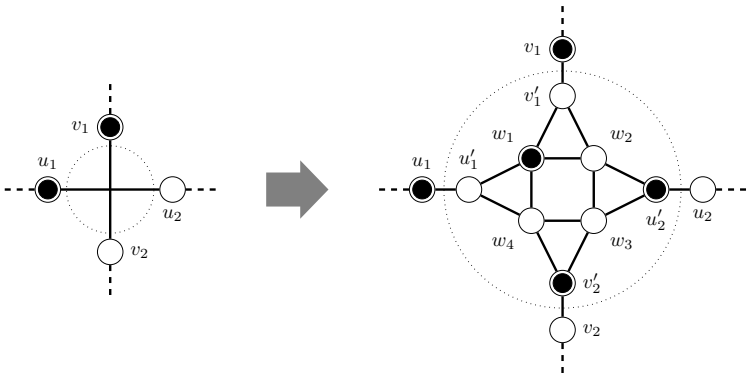


Figure 2 An illustration of replacing the crossing edges u_1u_2 and v_1v_2 with a crossover gadget. The gadget contains eight vertices and three tokens, and its maximum degree is 4.

$v_1'w_2$, $v_2'w_3$, $v_2'w_4$, w_1w_2, w_2w_3, w_3w_4, and w_4w_1. For two crossing edges u_1u_2 and v_1v_2 of G, *replacing* this crossing with a crossover gadget means removing the edges u_1u_2 and v_1v_2, inserting a crossover gadget, and adding the edges u_1u_1', u_2u_2', v_1v_1', and v_2v_2'. The crossover gadget always contains exactly three tokens corresponding to a maximum independent set of the gadget.

▶ **Theorem 3.** *Let μ be any fixed positive integer.* ISR *under k-*TJ *is* NP*-hard for planar graphs G of maximum degree 4 when $k = |I| - \mu \geq 1$, where I is an initial independent set of G.*

Proof. As the same reason in the proof of Theorem 2, we provide the proof when $\mu = 1$ (that is, $k = |I| - 1$). Let $(G, I, J, k\text{-TJ})$ be the instance of ISR constructed from ϕ' in the proof of Theorem 2, where $k = |I| - 1$. Consider any drawing of G on the plane, which may have crossings.

To make G into a planar graph of maximum degree 4, we replace each crossing in the drawing with a crossover gadget, which is shown in Figure 2. Suppose that we replaced a crossing of edges u_1u_2 and v_1v_2 with a crossover gadget. We claim that the crossover gadget correctly "simulates" the crossing if the crossing is *good*, that is, every independent set I' of G with size $|I|$ satisfies $|I' \cap \{u_1, u_2\}| = |I' \cap \{v_1, v_2\}| = 1$. Although not every drawing of G meets this condition[1], we claim that G always admits a drawing where all crossings are good. To this end, we construct a drawing of G in which all edge crossings occur only between edges belonging to distinct variable gadgets.

Let m be the number of clauses and n be the number of variables in ϕ', respectively. We will represent all variable gadgets and clause gadgets on a grid with $(2n + 3)$ rows and $6m$ columns such that edges between variable gadgets and clause gadgets have no crossing (see also Figure 3). Let (i, j) be the coordinates of a point on the Euclidean plane, where $i \in [6m]$ and $j \in [2n + 3]$. For each clause gadget corresponding to a clause C_i of ϕ', the three literal vertices of the gadget are positioned at $(6(i - 1) + 1, 2)$, $(6(i - 1) + 3, 2)$, and $(6(i - 1) + 5, 2)$. Furthermore, the edges are embedded to minimize the sum of their length, with one of the three edges utilizing the bottom border. If a literal vertex, corresponding to

[1] For example, the drawing in Figure 1 contains a non-good crossing of two edges $f^2_{x_3}v_{x_3}$ and $f^2_{x_4}u_{x_4}$, where the two vertices v_{x_3} and u_{x_4} correspond to the literal x_3 in clause C_3 and the literal x_4 in clause C_2, respectively. In this crossing, $f^2_{x_3}$ and v_{x_3} are both not in the independent set J.

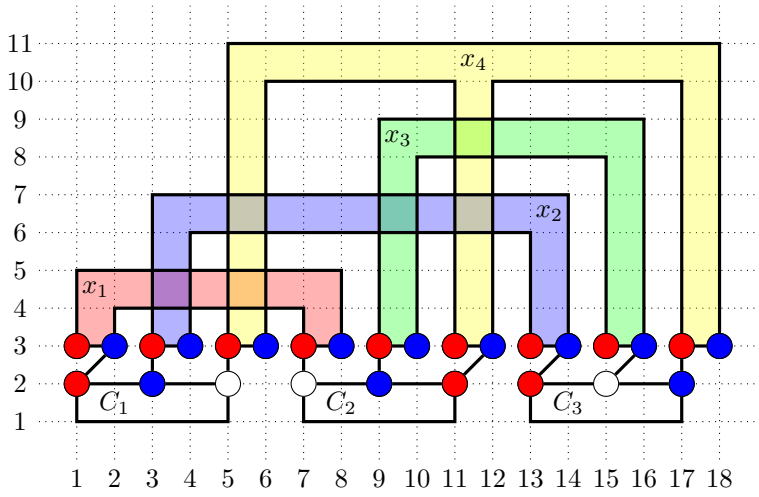


Figure 3 Another drawing of G (shown in Figure 1), corresponding to the Boolean formula $\phi' = (x_1 \vee \overline{x_2} \vee \overline{x_4}) \wedge (\overline{x_1} \vee \overline{x_3} \vee x_4) \wedge (x_2 \vee x_3 \vee \overline{x_4})$, on a grid with 11 rows and 18 columns. The intersection of a dotted horizontal line labeled $i \in \{1, \ldots, 11\}$ and a dotted vertical line labeled $j \in \{1, \ldots, 18\}$ represents the coordinate (i, j). Thick lines represent the edges of the graph G, and all edge crossings occur only between edges belonging to distinct variable gadgets.

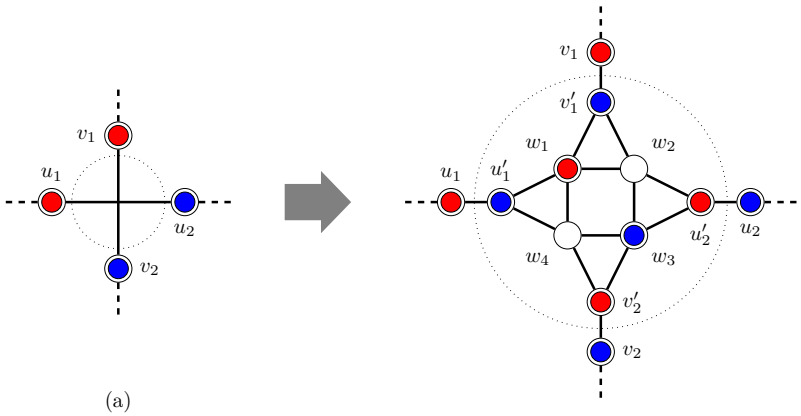
the j-th occurrence of a literal of variable x, is placed at $(i', 2)$, then the true vertex t_x^j and the false vertex f_x^j are positioned at $(i', 3)$ and $(i' + 1, 3)$, respectively, and are connected by a shortest edge.

Additionally, each literal vertex is joined with either of them, which follows the construction of G in the proof of Theorem 2: for a vertex v in a clause gadget corresponding to a literal l of a variable x with occurrence $j \in [a_x]$, connect v to t_x^j if l is negative, and to f_x^j otherwise. Then, for a variable gadget corresponding to a variable x_p with $p \in [n]$, we embed the remaining edges in the gadget using only the vertical lines where its vertices are located and the $(2p + 2)$-th and $(2p + 3)$-th horizontal lines, minimizing the total length. (If the variable gadget is a path with 2 vertices, there is nothing to do.)

Now, a new embedding of G is obtained and denoted by $D(G)$. Although $D(G)$ and G are isomorphic, $D(G)$ only has crossing edges in variable gadgets. Since every variable gadget forms a cycle of even length and has tokens placed alternately on its vertices, exactly one endpoint of each edge in the variable gadgets belongs to any independent set of G of size exactly $|I|$. Thus, all crossings of $D(G)$ are good.

We pick a crossing of two edges, say u_1u_2 and v_1v_2. Since both u_1u_2 and v_1v_2 are edges of variable gadgets, we may assume without loss of generality that $u_1, v_1 \in I$ and $u_2, v_2 \in J$. We then replace this crossing with our crossover gadget. Let G^* denote the resulting graph. For the initial (resp. target) independent set I (resp. J) of G, adding three tokens on w_1, u_2', and v_2' (resp. w_3, u_1', and v_1') in the crossover gadget (as shown in Figure 4) results in the independent set I^* (resp. J^*). Let $(G^*, I^*, J^*, k\text{-}\mathsf{TJ})$ be a new instance of ISR under k-TJ, where $k = |I^*| - 1$. Note that, since the sets of vertices added to I and J are disjoint, we have $I^* \cap J^* = \emptyset$.

For each configuration of tokens on u_1, u_2, v_1, and v_2 in G, there exists a unique configuration of three tokens on the crossover gadgets in G^* as shown in Figure 5. Thus, the configuration of tokens on the vertices of the crossover gadget can be changed in G^* if and only if the configuration of tokens on u_1, u_2, v_1, and v_2 is changed in G. Therefore,



(a) (b)

Figure 4 An illustration of adding tokens to I and J, yielding I^* and J^*. (a) Vertices u_1 and v_1 belong to I (red tokens), and u_2 and v_2 belong to J (blue tokens). (b) In I^*, the new vertices w_1, u_2', and v_2' are added, while in J^*, the new vertices w_3, u_1', and v_1' are added. The two independent sets I^* and J^* remain disjoint.

by faithfully simulating token moves along the edges u_1u_2 and v_1v_2 of G using the token arrangements shown in Figure 5, one can show that $(G, I, J, k\text{-TJ})$ is a yes-instance if and only if $(G^*, I^*, J^*, k\text{-TJ})$ is a yes-instance.

The graph G^* has one less edge crossing than G. Consequently, we can obtain a plane graph G' by repeating the above replacement for each crossing of $D(G)$. As $D(G)$ has $O(mn)$ crossings, the construction of G' can be done in polynomial time. Since a crossover gadget is a graph of maximum degree 4, G' is a plane graph of maximum degree 4. This completes our proof. ◀

3.1.1.1 Proof of Lemma 1

Proof. It is obvious that IntE3-SAT is in NP. To prove the NP-completeness, we use a polynomial-time reduction from E3-SAT.

Let $\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ be an instance of E3-SAT, and $X = \{x_1, \ldots, x_n\}$ be the variable set of ϕ. We will construct a sandwiched E3-CNF formula ϕ' with $7mn^2$ clauses and $n+2mn^2$ variables at the end of our construction. Before starting our construction, we restrict ϕ so that neither the all-T assignment nor the all-F assignment satisfies ϕ. E3-SAT remains NP-hard with this restriction; otherwise, we could solve E3-SAT without the restriction by checking whether the all-T or all-F assignment satisfies ϕ at first. We obtain a new CNF formula ϕ^* as follows:

$$\phi^* = \phi \vee \bigwedge_{i=1}^{n} x_i \vee \bigwedge_{i=1}^{n} \overline{x_i} = \bigwedge_{h=1}^{m} C_h \vee \bigwedge_{i=1}^{n} x_i \vee \bigwedge_{i=1}^{n} \overline{x_i} = \bigwedge_{h=1}^{m} \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n} (C_h \vee x_i \vee \overline{x_j}).$$

Since C_h consists of exactly three literals, $C_h \vee x_i \vee \overline{x_j}$ is a disjunction of five literals. Thus, ϕ^* is an E5-CNF formula. Furthermore, ϕ^* is a sandwiched E5-CNF formula, that is, each clause of ϕ^* has both a positive literal x_i and a negative literal $\overline{x_j}$. From our conversion, ϕ^* has mn^2 clauses and n variables. It is observed that for any mixed variable assignment b, ϕ evaluates T if and only if ϕ^* evaluates T. Thus, we can immediately say that ϕ is a yes-instance of E3-SAT without all-T and all-F assignments if and only if ϕ' is a yes-instance of IntE5-SAT.

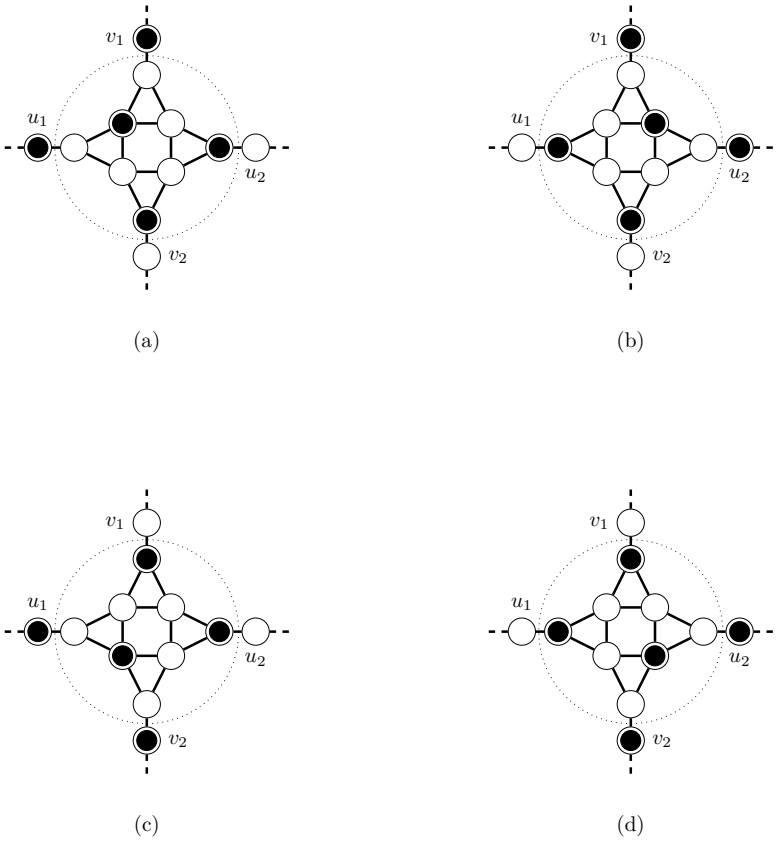


Figure 5 An illustration of how the crossover gadget works. The crossing edges u_1u_2 and v_1v_2, where one endpoint of each edge is occupied by a token, are replaced by a crossover gadget: (a) u_1 and v_1 are occupied; (b) u_2 and v_1 are occupied; (c) u_1 and v_2 are occupied; (b) u_2 and v_2 are occupied.

Then, we explain how to convert a sandwiched E5-CNF formula ϕ^* to a sandwiched E3-CNF formula ϕ' with $7mn^2$ clauses and $n + 2mn^2$ variables. We repeat the following operation on ϕ^* until all clauses have size exactly 3. Let $\phi_0^* = \phi^*$ and ϕ_i^* for a positive integer i be a formula obtained from a sandwiched CNF formula ϕ_{i-1}^*.

Let C be a clause in ϕ_{i-1}^* with size at least 4. Then C contains $x \vee y$ or $\overline{x} \vee \overline{y}$ for variables x, y. If C contains $x \vee y$, then we replace $x \vee y$ with a new variable z and combine the modified formula with the formula $x \vee y \leftrightarrow z$ using the $\wedge$ operator. Furthermore, $x \vee y \leftrightarrow z$ is transformed as follows:

$$\begin{aligned} x \vee y \leftrightarrow z &= (x \vee y \vee \overline{z}) \wedge (\overline{(x \vee y)} \vee z) \\ &= (x \vee y \vee \overline{z}) \wedge ((\overline{x} \wedge \overline{y}) \vee z) \\ &= (x \vee y \vee \overline{z}) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee z) \\ &= (x \vee y \vee \overline{z}) \wedge (\overline{x} \vee \overline{x} \vee z) \wedge (\overline{y} \vee \overline{y} \vee z). \end{aligned} \tag{1}$$

We claim that ϕ_i^* obtained from ϕ_{i-1}^* by the above operation is a sandwiched CNF formula. Clearly, each clause in ϕ_i^* that remains unchanged from ϕ_{i-1}^* contains both positive and negative literals. The clause in ϕ_i^* obtained from C by replacing $x \vee y$ with a positive literal z contains a negative literal because C also has a negative literal. Combined with Equation (1), ϕ_i^* is a sandwiched CNF formula. Consequently, if there is a clause in ϕ_{i-1}^* that contains $x \vee y$, then a sandwiched CNF formula ϕ_i^* is obtained.

Similarly, if C contains $\overline{x} \vee \overline{y}$, then we replace $\overline{x} \vee \overline{y}$ with the negative literal of a new variable z and combine the modified formula with the formula $\overline{x} \vee \overline{y} \leftrightarrow \overline{z} = (\overline{x} \vee \overline{y} \vee z) \wedge (x \vee x \vee \overline{z}) \wedge (y \vee y \vee \overline{z})$ using the $\wedge$ operator. As with the previous argument, we can say that ϕ_i^* is a sandwiched CNF formula.

Let ϕ' be the sandwiched CNF formula obtained from $\phi^* = \phi_0^*$ by repeating the above operation until all clauses have size exactly 3. Since exactly two replacements occur per clause in ϕ^*, we have $\phi' = \phi_{2mn^2}^*$. Moreover, six new clauses and two new variables are added for each clause in ϕ^*. Therefore, ϕ' has $7mn^2$ clauses and $n + 2mn^2$ variables.

To complete our reduction, for each $i \in [2mn^2]$, we will show that there exists a mixed variable assignment that satisfies ϕ_{i-1}^* if and only if there exists a mixed variable assignment that satisfies ϕ_i^*. This immediately implies that $\phi^* = \phi_0^*$ is a yes-instance of IntE5-SAT if and only if $\phi' = \phi_{2mn^2}^*$ is a yes-instance of IntE3-SAT. Let X_i be the set of variables in ϕ_i^*.

Suppose that there is a mixed variable assignment b_{i-1} for X_{i-1} that satisfies ϕ_{i-1}^*. Consider a variable assignment b_i for $X_i = X_{i-1} \cup \{z\}$ such that $b_i(x) = b_{i-1}(x)$ for each $x \in X_{i-1}$. Moreover, set $b_i(z) = x \vee y$ if z replaces $x \vee y$, and set $b_i(z) = \overline{\overline{x} \vee \overline{y}}$ if $\overline{z}$ replaces $\overline{x} \vee \overline{y}$. Since b_{i-1} is a mixed variable assignment, b_i is also a mixed variable assignment. Furthermore, it is easy to see that b_i satisfies ϕ_i^*.

Conversely, suppose that there is a mixed variable assignment b_i for X_i that satisfies ϕ_i^*. Due to $x \vee y \leftrightarrow z$ or $\overline{x} \vee \overline{y} \leftrightarrow \overline{z}$, the variable assignment b_{i-1} such that $b_{i-1}(x) = b_i(x)$ for each $x \in X_{i-1}$ satisfies ϕ_{i-1}^*. We claim that the variable assignment b_{i-1} is mixed. For the sake of contradiction, assume that b_{i-1} is not mixed, that is, b_{i-1} is either the all-T assignment or all-F assignment. If $x \vee y \leftrightarrow z$ is added into ϕ_{i-1}^*, then $b_i(x) = b_i(y) = b_i(z)$ holds. Thus, b_i is also not mixed, a contradiction. Similarly, if $\overline{x} \vee \overline{y} \leftrightarrow \overline{z}$ is added into ϕ_{i-1}^*, then $b_i(x) = b_i(y) = b_i(z)$ holds, a contradiction. Therefore, we conclude that b_{i-1} is a mixed variable assignment for X_{i-1} that satisfies ϕ_{i-1}^*. This completes the proof. ◀

3.1.2 Membership in NP

Next, we show that ISR under k-TJ with $k = |I| - \mu$ is in NP not only when μ is constant but also μ is at most $O(\log |I|)$ for graphs of bounded maximum degree and planar graphs of maximum degree $o(\frac{n}{\log n})$, where n is the number of vertices in the input graph.

▶ **Theorem 4.** *Let G be an input graph with n vertices, chromatic number $O(1)$ and maximum degree $o(\frac{n}{\log n})$, and let I be an initial independent set of G.* ISR *under k-TJ is in* NP *when $k = |I| - \mu \geq 1$ with any non-negative integer μ at most $O(\log |I|)$.*

To prove Theorem 4, we will evaluate the length of a shortest reconfiguration sequence between any two independent sets of an input graph. Firstly, we introduce a lemma on the maximum size of intersecting families of sets. Let N, r be positive integers with $N \geq r$, and let $L \subseteq \{0, 1, \ldots, r-1\}$. We say that a family $\mathcal{F}$ of r-element subsets of $[N]$ is an (N, r, L)-*system* if $|F \cap F'| \in L$ holds for all distinct $F, F' \in \mathcal{F}$. Let $m(N, r, L)$ denote the maximum size of (N, r, L)-systems.

For any positive integers N, r, p with $N \geq r > p \geq 1$, the following upper bound is known (see, for example, [14, 25, 33]):

$$m(N, r, \{0, \ldots, p-1\}) \leq \tbinom{N}{p} / \tbinom{r}{p}. \tag{2}$$

Using Equation (2), we prove Lemma 5, which provides an upper bound on the length of any shortest reconfiguration sequence in the general case.

▶ **Lemma 5.** *Let G be an input graph with $n = |V(G)|$, I and J be initial and target independent sets of G, and $\mu < |I|$ be any non-negative integer. If I and J are reconfigurable under k-TJ, then the length of a shortest reconfiguration sequence between I and J under k-TJ, where $k = |I| - \mu \geq 1$, is at most $O((\frac{n}{k})^{\mu})$.*

Proof. When $\mu = 0$, there exists a reconfiguration sequence $\langle I, J\rangle$ of length 1. Therefore, we assume that $\mu \geq 1$. Consider any shortest reconfiguration sequence $\sigma = \langle I = I_0, I_1, \ldots, I_\ell = J\rangle$ between I and J. Let $\mathcal{I} = \{I_i \colon i \in \{0, 1, \ldots, \ell\}, i \text{ is even}\}$. Note that $|\mathcal{I}| = \lfloor \frac{\ell}{2} \rfloor + 1$. Since σ is the shortest reconfiguration sequence, for any two independent sets I_i and I_j in $\mathcal{I}$ with $i < j$, we have $|I_i \cap I_j| < \mu$; otherwise, I_i and I_j are adjacent under k-TJ and we can obtain a shorter reconfiguration sequence σ' from σ by removing all independent sets $I_{i+1}, \ldots, I_{j-1}$, which is a contradiction. Then, we observe that $\mathcal{I}$ is an $(n, |I|, \{0, \ldots, \mu - 1\})$-system. By using Equation (2), we have

$$|\mathcal{I}| = \left\lfloor \frac{\ell}{2} \right\rfloor + 1 \leq m(n, |I|, \{0, \ldots, \mu - 1\}) \leq \binom{n}{\mu} / \binom{|I|}{\mu} \leq \left(\frac{n}{|I| - \mu}\right)^{\mu}.$$

Therefore, the length ℓ of σ is at most $O((\frac{n}{|I|-\mu})^{\mu}) = O((\frac{n}{k})^{\mu})$. ◀

Now, we can prove Theorem 4.

Proof of Theorem 4. It is trivial when $\mu = 0$, thus we assume that $\mu \geq 1$. Suppose first that $2\mu \geq |I|$. Then, there is some constant c such that $|I| < c$ since $\mu = O(\log |I|)$. We can solve ISR under k-TJ by enumerating all independent sets of G with constant size in polynomial time.

Suppose next that $2\mu < |I|$ for sufficiently large $|I|$. Let A and B be two independent sets such that $A \subseteq I$ and $B \subseteq J$ with size exactly μ. Let G^* be the subgraph of G obtained by removing all vertices in $N[A \cup B]$. If G^* has an independent set I^* with size k, then there is a reconfiguration sequence $\langle I, A \cup I^*, B \cup I^*, J\rangle$ between I and J. We say that this reconfiguration sequence is a *simple reconfiguration sequence*. Since the vertex set of G^* is $V(G) \setminus N[A \cup B]$, the size of $V(G^*)$ is at least $n - 2\mu(\Delta + 1)$, where Δ is the maximum degree of G. Let χ and χ^* be the chromatic numbers of G and G^*, respectively. Then, we observe that $\chi^* \leq \chi$. By the relationship between the chromatic number and the independence number, G^* has an independent set with size at least $|V(G^*)|/\chi^* \geq (n - 2\mu(\Delta + 1))/\chi^*$. Thus, if $(n - 2\mu(\Delta + 1))/\chi^* \geq k$, then I and J are always reconfigurable, as a simple reconfiguration sequence exists. Note that the length of a simple reconfiguration sequence is 3.

It remains to consider the case where $(n - 2\mu(\Delta + 1))/\chi^* < k$, in which a simple reconfiguration sequence between I and J may not exist. Combined with Lemma 5, the length ℓ of a shortest reconfiguration sequence between I and J satisfies

$$\ell = O\left(\left(\frac{n}{k}\right)^{\mu}\right) = O\left(\left(\frac{n\chi^*}{n - 2\mu(\Delta + 1)}\right)^{\mu}\right) = O\left((\chi^*)^{\mu}\left(\frac{1}{1 - \frac{2\mu(\Delta+1)}{n}}\right)^{\mu}\right). \tag{3}$$

Since $\mu = O(\log |I|) = O(\log n)$ and $\Delta = o(\frac{n}{\log n})$, we have $\mu\Delta = o(n)$. Hence, we have $(2\mu(\Delta + 1))/n = o(1)$. Furthermore, $\chi^* \leq \chi = O(1)$. Thus, from Equation (3), we have

$$\ell = O((\chi^*)^{\mu}(\frac{1}{1 - o(1)})^{\mu}) = O(1)^{O(\log n)} = O(n^{O(1)}).$$

Therefore, ℓ is polynomially bounded in n, and hence the problem belongs to NP. This completes the proof. ◀

It is known that the chromatic number of G is at most $\Delta + 1$, where Δ is the maximum degree of G [8]. In addition, the chromatic number of any planar graph is at most 4 [1, 2]. Therefore, Theorem 4 gives the results including graphs of bounded maximum degree and planar graphs of maximum degree $o(\frac{n}{\log n})$.

3.2 VCR

In Section 3.1, we showed that ISR under k-TJ with $k = |I| - \mu$, where μ is any fixed positive integer, is NP-complete even for graphs of maximum degree 3 and for planar graphs of maximum degree 4. In contrast to this intractability, VCR under k-TJ is in XP for general graphs when parameterized by $\mu = |S| - k > 0$, where S is an initial vertex cover of an input graph.

▶ **Theorem 6** (∗). VCR *under* k-TJ *is in* XP *for general graphs* G *when parameterized by* $\mu = |S| - k \geq 0$*, where* S *is an initial vertex cover of* G*.*

In the proof of Theorem 6, we present an XP algorithm for the problem.

Let $(G, S, T, k\text{-TJ})$ be an instance of VCR. We can consider the *reconfiguration graph* $\mathcal{C} = (\mathcal{V}, \mathcal{E})$ for the instance, such that each node $w_{S'} \in \mathcal{V}$ corresponds to a vertex cover S' of G with size exactly $|S|$, and edges represent adjacency under k-TJ. Since the number of such vertex covers can be superpolynomial, explicitly constructing $\mathcal{C}$ is infeasible in general.

Our approach builds on the clique-compressed reconfiguration graph technique introduced in [35], which compactly represents $\mathcal{C}$ by grouping cliques into single nodes. This compressed graph has at most $O(n^{\mu})$ nodes and preserves essential connectivity, making it sufficient for solving the problem if constructed efficiently.

Here, we briefly describe the characteristics of instances that enable the construction of our XP algorithm. If $|S \cap T| \geq \mu$, then S and T are reconfigurable, since we can simultaneously move $k = |S| - \mu$ tokens. Therefore, we focus on the case where $|S \cap T| < \mu$. Since both S and T are vertex covers, the induced subgraph $G[V(G) \setminus (S \cap T)]$ is a bipartite graph. This restricted structural property serves as a key ingredient in the design of our XP algorithm.

4 PSPACE-completeness When k is Constant

In this section, we investigate the PSPACE-completeness of ISR under k-TJ and k-TS when k is fixed. Note that the computational complexity of ISR and VCR under k-TJ and k-TS is the same when k is fixed, due to their complementary relationship.

4.1 Planar Graphs

▶ **Theorem 7** (∗). *Let* $k \geq 2$ *be any fixed positive integer.* ISR *under* $\mathsf{R} \in \{k\text{-TS}, k\text{-TJ}\}$ *is* PSPACE*-complete for planar graphs of maximum degree* 3 *and bounded bandwidth.*

To prove Theorem 7, we construct a reduction from Nondeterministic Constraint Logic, which was invented by Hearn and Demaine [18] and has been used to prove the PSPACE-hardness of reconfiguration problems, including ISR under TS [18].

4.2 Line Graphs and Claw-free Graphs

We state that ISR under $\mathsf{R} \in \{2\text{-TJ}, 2\text{-TS}\}$ is PSPACE-complete even for line graphs, which contrasts that ISR under $\mathsf{R} \in \{\mathsf{TJ}, \mathsf{TS}\}$ can be solved in polynomial time for line graphs (more generally, claw-free graphs) [6]. For a graph G, its *line graph* $L(G)$ is defined as follows: each vertex of $L(G)$ corresponds to an edge of G, and two vertices in $L(G)$ are adjacent if and only if their corresponding edges in G share a common endpoint.

▶ **Theorem 8** (∗). ISR *under* R ∈ {2-TJ, 2-TS} *is* PSPACE*-complete for line graphs.*

In the proof of Theorem 8, we reduce Perfect Matching Reconfiguration, which is known to be PSPACE-complete on bipartite graphs of maximum degree 5 and bounded bandwidth [5], to our problem.

5 PSPACE-completeness When k is Superconstant

This section is devoted to establishing the PSPACE-completeness of ISR and VCR under k-TJ when k is superconstant in the initial independent set size $|I|$ and the initial vertex cover size $|S|$, respectively.

5.1 ISR

The main result in this section is the following.

▶ **Theorem 9.** *There exists some constant* $\varepsilon_0 \in (0,1)$ *such that* ISR *under* k-TJ *on graphs of maximum degree* 3 *is* PSPACE*-complete for any* k *satisfying the following condition: there exists a constant* c *such that* $k \leq \varepsilon_0 |I|$ *holds whenever* $|I| \geq c$*, where* I *is the initial independent set of the input graph.*

To prove Theorem 9, we will construct a polynomial-time reduction from the *optimization variant* of ISR called Maxmin Independent Set Reconfiguration (MaxminISR for short) [20]. In the problem, we adopt the *token addition and removal* (TAR for short) [20], under which two independent sets of a graph G are adjacent if one is obtained from the other by adding or removing a single vertex of G. In MaxminISR, given a graph G and two independent sets I, J of G, we are asked to find a reconfiguration sequence $\sigma = \langle I = I_0, I_1, \ldots, I_\ell = J \rangle$ under TAR that maximizes $\mathsf{val}(\sigma)$, where $\mathsf{val}(\sigma) = \min\{|I'| \colon I' \in \sigma\}$. For a graph G, we use $\alpha(G)$ to denote the number of vertices in a maximum independent set of G. Let (G, I, J) be an instance of MaxminISR, and $\mathsf{val}_{\mathsf{max}}(I, J)$ be the maximum value of $\mathsf{val}(\sigma)$ over all possible reconfiguration sequences σ from I to J under TAR. Recently, the following Theorem 10 was proven [19, 24, 32].

▶ **Theorem 10** ([19, 24, 32])**.** *Let* I *and* J *be initial and target independent sets of an input graph* G *in* MaxminISR*. Then, there exists some constant* $\varepsilon_0 \in (0,1)$ *such that it is* PSPACE*-hard to distinguish between the following two cases:*
(i) $\mathsf{val}_{\mathsf{max}}(I, J) \geq \alpha(G) - 1$*, and*
(ii) $\mathsf{val}_{\mathsf{max}}(I, J) < (1 - \varepsilon_0)(\alpha(G) - 1)$*.*
The same hardness result holds even when the maximum degree of G *is* 3*,* $|I| = |J| = \alpha(G)$*, and* $\frac{\alpha(G)}{|V(G)|} \in [\frac{1}{3}, \frac{1}{2}]$*.*

To lead to Theorem 9 from Theorem 10, we provide the following lemma.

▶ **Lemma 11.** *Let* I *and* J *be initial and target independent sets of an input graph* G *in* MaxminISR *and* ISR*. Let* f *be a given function and* g *be any function defined on integers such that* $x - g(x) \geq 1$ *for all positive integers* x *and there exists a fixed positive integer* n_0 *satisfying* $g(n) \geq f(n)$ *for all integers* $n \geq n_0$*. Suppose that it is* PSPACE*-hard to distinguish between the following two cases:*
(i) $\mathsf{val}_{\mathsf{max}}(I, J) \geq |I| - 1$*, and*
(ii) $\mathsf{val}_{\mathsf{max}}(I, J) < f(|I|)$*.*
Then, ISR *under* k-TJ *is* PSPACE*-hard, where* $k = |I| - g(|I|) \geq 1$*.*

Proof. Let (G, I, J) be an instance of MaxminISR, and $(G, I, J, k\text{-TJ})$ be an instance of ISR where $k = |I| - g(|I|)$. We assume that $|I| \geq n_0$ and hence $k \geq 1$ since we can solve all instances with $|I| < n_0$ by enumerating all independent sets of constant size. We now show that if (G, I, J) satisfies condition (i), then $(G, I, J, k\text{-TJ})$ is a yes-instance, and if (G, I, J) satisfies condition (ii), then $(G, I, J, k\text{-TJ})$ is a no-instance. We will show the latter one by proving the contrapositive: if $(G, I, J, k\text{-TJ})$ is a yes-instance, then (G, I, J) does not satisfy condition (ii).

Firstly, suppose that there is a reconfiguration sequence $\sigma = \langle I = I_0, I_1, \ldots, I_\ell = J \rangle$ between I and J under TAR such that $\mathsf{val}(\sigma) \geq |I| - 1$. It is known that this assumption holds if and only if there is a reconfiguration sequence under TJ between I and J [23]. Thus, there is a reconfiguration sequence under TJ between I and J, and that is also a reconfiguration sequence under k-TJ. Therefore, $(G, I, J, k\text{-TJ})$ is a yes-instance.

Conversely, suppose that there is a reconfiguration sequence $\sigma' = \langle I = I_0, I_1, \ldots, I_\ell = J \rangle$ between I and J under k-TJ where $k = |I| - g(|I|) \geq 1$. Then, for any two consecutive independent sets I_{i-1} and I_i with $i \in [\ell]$, we have $|I_{i-1} \cap I_i| \geq |I| - k = g(|I|)$. Additionally, we can transform from I_{i-1} to I_i under TAR as follows: Firstly, we remove tokens on vertices in $I_{i-1} \setminus I_i$ one by one; then, we add tokens on vertices in $I_i \setminus I_{i-1}$ one by one. Through these steps, we have no independent set with size smaller than $|I_{i-1} \cap I_i| \geq g(|I|) \geq f(|I|)$. Therefore, there is a sequence σ under TAR such that $\mathsf{val}(\sigma) \geq g(|I|) \geq f(|I|)$. That is, (G, I, J) does not satisfy condition (ii). This completes the proof. ◀

We set $f(x) = (1 - \varepsilon_0)(x - 1)$ and let $g(x)$ be an arbitrary function such that $x - 1 \geq g(x) \geq f(x)$ for all $x \geq x_0$, for some constant x_0. For example, $g(x)$ may be chosen as $x - c$ for some constant c, $x - \lceil \log x \rceil$, $x - \lceil x^{1/2} \rceil$, or $x - \lceil \varepsilon x \rceil$ for some constant $\varepsilon \leq \varepsilon_0$. Combining Theorem 10 and Lemma 11, ISR under k-TJ on graphs of maximum degree 3 is PSPACE-complete for $k = |I| - g(|I|)$, as claimed in Theorem 9. This result includes the PSPACE-completeness of ISR under k-TJ for various values of k, such as $\Theta(1)$, $\Theta(\log |I|)$, $\Theta(|I|^{O(1)})$, and $\Theta(|I|)$.

5.2 VCR

Similarly to Theorem 9, we can prove the following Theorem 12.

▶ **Theorem 12** (∗). *There exists some constant $\varepsilon_0 \in (0, 1)$ such that* VCR *under k-TJ on graphs of maximum degree* 3 *is* PSPACE*-complete for any k satisfying the following condition: there exists a constant c such that $k \leq \varepsilon_0 |S|$ holds whenever $|S| \geq c$, where S is the initial vertex cover of the input graph.*

6 Conclusion and Future Work

In this paper, we investigated the computational complexity of the fundamental reconfiguration problems ISR and VCR on various graph classes under the extended reconfiguration rules k-TJ and k-TS.

The following open problems are suggested for future research: (1) Is ISR under k-TJ with $k = |I| - \mu$ NP-hard on planar graphs of maximum degree 3 for any fixed positive integer μ? (2) Is ISR under k-TJ with $k = |I| - \mu$ in NP on general graphs not only when μ is fixed but also when $\mu = O(\log |I|)$? (See also Table 1.)

References

1 Kenneth Appel and Wolfgang Haken. Every planar map is four colorable. part I: Discharging. *Illinois J. Math.*, 21(3):429–490, 1977. doi:10.1215/ijm/1256049011.

2 Kenneth Appel, Wolfgang Haken, and John Koch. Every planar map is four colorable. part II: Reducibility. *Illinois J. Math.*, 21(3):491–567, 1977. doi:10.1215/ijm/1256049012.

3 Rémy Belmonte, Eun Jung Kim, Michael Lampis, Valia Mitsou, Yota Otachi, and Florian Sikora. Token sliding on split graphs. *Theory Comput. Syst.*, 65(4):662–686, 2021. doi:10.1007/S00224-020-09967-8.

4 Marthe Bonamy and Nicolas Bousquet. Token sliding on chordal graphs. In *WG*, pages 127–139, 2017. doi:10.1007/978-3-319-68705-6_10.

5 Marthe Bonamy, Nicolas Bousquet, Marc Heinrich, Takehiro Ito, Yusuke Kobayashi, Arnaud Mary, Moritz Mühlenthaler, and Kunihiro Wasa. The perfect matching reconfiguration problem. In *MFCS*, pages 80:1–80:14, 2019. doi:10.4230/LIPICS.MFCS.2019.80.

6 Paul S. Bonsma, Marcin Kamiński, and Marcin Wrochna. Reconfiguring independent sets in claw-free graphs. In *SWAT*, pages 86–97, 2014. doi:10.1007/978-3-319-08404-6_8.

7 Marcin Briański, Stefan Felsner, Jędrzej Hodor, and Piotr Micek. Reconfiguring independent sets on interval graphs. In *MFCS*, pages 23:1–23:14, 2021. doi:10.4230/LIPICS.MFCS.2021.23.

8 R. L. Brooks. On colouring the nodes of a network. *Mathematical Proceedings of the Cambridge Philosophical Society*, 37(2):194–197, 1941. doi:10.1017/S030500410002168X.

9 Remo Christen, Salomé Eriksson, Michael Katz, Christian Muise, Alice Petrov, Florian Pommerening, Jendrik Seipp, Silvan Sievers, and David Speck. PARIS: planning algorithms for reconfiguring independent sets. In *ECAI*, pages 453–460, 2023. doi:10.3233/FAIA230303.

10 Mark de Berg, Bart M. P. Jansen, and Debankur Mukherjee. Independent-set reconfiguration thresholds of hereditary graph classes. *Discret. Appl. Math.*, 250:165–182, 2018. doi:10.1016/J.DAM.2018.05.029.

11 Erik D. Demaine, Martin L. Demaine, Eli Fox-Epstein, Duc A. Hoang, Takehiro Ito, Hirotaka Ono, Yota Otachi, Ryuhei Uehara, and Takeshi Yamada. Linear-time algorithm for sliding tokens on trees. *Theor. Comput. Sci.*, 600:132–142, 2015. doi:10.1016/J.TCS.2015.07.037.

12 Naoki Domon, Akira Suzuki, Yuma Tamura, and Xiao Zhou. The shortest path reconfiguration problem based on relaxation of reconfiguration rules. In *WALCOM*, pages 227–241, 2024. doi:10.1007/978-981-97-0566-5_17.

13 Paul Erdős, Chao Ko, and Richard Rado. Intersection theorems for systems of finite sets. *Q. J. Math.*, 12(1):313–320, 1961. doi:10.1093/qmath/12.1.313.

14 Peter Frankl and Norihide Tokushige. Invitation to intersection problems for finite sets. *J. Comb. Theory A*, 144:157–211, 2016. doi:10.1016/J.JCTA.2016.06.017.

15 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

16 Arash Haddadan, Takehiro Ito, Amer E. Mouawad, Naomi Nishimura, Hirotaka Ono, Akira Suzuki, and Youcef Tebbal. The complexity of dominating set reconfiguration. *Theor. Comput. Sci.*, 651:37–49, 2016. doi:10.1016/J.TCS.2016.08.016.

17 Hiroki Hatano, Naoki Kitamura, Taisuke Izumi, Takehiro Ito, and Toshimitsu Masuzawa. Independent set reconfiguration under bounded-hop token jumping. In *WALCOM*, pages 215–228, 2025. doi:10.1007/978-981-96-2845-2_14.

18 Robert A. Hearn and Erik D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theor. Comput. Sci.*, 343(1-2):72–96, 2005. doi:10.1016/J.TCS.2005.05.008.

19 Shuichi Hirahara and Naoto Ohsaka. Probabilistically checkable reconfiguration proofs and inapproximability of reconfiguration problems. In *STOC*, pages 1435–1445, 2024. doi:10.1145/3618260.3649667.

20 Takehiro Ito, Erik D. Demaine, Nicholas J. A. Harvey, Christos H. Papadimitriou, Martha Sideri, Ryuhei Uehara, and Yushi Uno. On the complexity of reconfiguration problems. *Theor. Comput. Sci.*, 412(12-14):1054–1065, 2011. doi:10.1016/J.TCS.2010.12.005.

21 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 *CPAIOR*, pages 167–183, 2023. doi:10.1007/978-3-031-33271-5_12.

22 Takehiro Ito, Hirotaka Ono, and Yota Otachi. Reconfiguration of cliques in a graph. *Discret. Appl. Math.*, 333:43–58, 2023. doi:10.1016/J.DAM.2023.01.026.

23 Marcin Kamiński, Paul Medvedev, and Martin Milanič. Complexity of independent set reconfigurability problems. *Theor. Comput. Sci.*, 439:9–15, 2012. doi:10.1016/J.TCS.2012.03.004.

24 Karthik C. S. and Pasin Manurangsi. On inapproximability of reconfiguration problems: PSPACE-hardness and some tight NP-hardness results. *CoRR*, abs/2312.17140, 2023. doi:10.48550/arXiv.2312.17140.

25 Gyula Katona, Tibor Nemetz, and Miklos Simonovits. On a graph-problem of Turán. *Matematikai Lapok*, 15, 1964.

26 Jan Matyáš Křišťan and Jakub Svoboda. Reconfiguration using generalized token jumping. In *WALCOM*, pages 244–265, 2025. doi:10.1007/978-981-96-2845-2_16.

27 Daniel Lokshtanov and Amer E. Mouawad. The complexity of independent set reconfiguration on bipartite graphs. *ACM Trans. Algorithms*, 15(1):7:1–7:19, 2019. doi:10.1145/3280825.

28 Meena Mahajan and Venkatesh Raman. Parameterizing above guaranteed values: MaxSat and MaxCut. *J. Algorithms*, 31(2):335–354, 1999. doi:10.1006/JAGM.1998.0996.

29 Amer E. Mouawad, Naomi Nishimura, Venkatesh Raman, and Sebastian Siebertz. Vertex cover reconfiguration and beyond. *Algorithms*, 11(2):20, 2018. doi:10.3390/A11020020.

30 Amer E. Mouawad, Naomi Nishimura, Venkatesh Raman, Narges Simjour, and Akira Suzuki. On the parameterized complexity of reconfiguration problems. *Algorithmica*, 78(1):274–297, 2017. doi:10.1007/S00453-016-0159-2.

31 Naomi Nishimura. Introduction to reconfiguration. *Algorithms*, 11(4):52, 2018. doi:10.3390/A11040052.

32 Naoto Ohsaka. Gap preserving reductions between reconfiguration problems. In *STACS*, pages 49:1–49:18, 2023. doi:10.4230/LIPICS.STACS.2023.49.

33 Vojtech Rödl. On a packing and covering problem. *Eur. J. Comb.*, 6(1):69–78, 1985. doi:10.1016/S0195-6698(85)80023-8.

34 Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970. doi:10.1016/S0022-0000(70)80006-X.

35 Tatsuhiro Suga, Akira Suzuki, Yuma Tamura, and Xiao Zhou. Changing induced subgraph isomorphisms under extended reconfiguration rules. In *WALCOM*, pages 346–360, 2025. doi:10.1007/978-981-96-2845-2_22.

36 Akira Suzuki, Amer E. Mouawad, and Naomi Nishimura. Reconfiguration of dominating sets. *J. Comb. Optim.*, 32(4):1182–1195, 2016. doi:10.1007/S10878-015-9947-X.

37 Takahisa Toda, Takehiro Ito, Jun Kawahara, Takehide Soh, Akira Suzuki, and Junichi Teruyama. Solving reconfiguration problems of first-order expressible properties of graph vertices with boolean satisfiability. In *ICTAI*, pages 294–302, 2023. doi:10.1109/ICTAI59109.2023.00050.

38 Jan van den Heuvel. The complexity of change. In *Surveys in Combinatorics 2013*, volume 409 of *London Mathematical Society Lecture Note Series*, pages 127–160. Cambridge University Press, 2013. doi:10.1017/CBO9781139506748.005.

39 Tom C. van der Zanden. Parameterized complexity of graph constraint logic. In *IPEC*, pages 282–293, 2015. doi:10.4230/LIPICS.IPEC.2015.282.

40 Marcin Wrochna. Reconfiguration in bounded bandwidth and tree-depth. *J. Comput. Syst. Sci.*, 93:1–10, 2018. doi:10.1016/J.JCSS.2017.11.003.

41 Yuya Yamada, Mutsunori Banbara, Katsumi Inoue, Torsten Schaub, and Ryuhei Uehara. Combinatorial reconfiguration with answer set programming: Algorithms, encodings, and empirical analysis. In *WALCOM*, pages 242–256, 2024. doi:10.1007/978-981-97-0566-5_18.

Minimum Sum Coloring with Bundles in Trees and Bipartite Graphs

Takehiro Ito
Graduate School of Information Sciences, Tohoku University, Sendai, Japan

Naonori Kakimura
Faculty of Science and Technology, Keio University, Yokohama, Japan

Naoyuki Kamiyama
Institute of Mathematics for Industry, Kyushu University, Fukuoka, Japan

Yusuke Kobayashi
Research Institute for Mathematical Sciences, Kyoto University, Japan

Yoshio Okamoto
Graduate School of Informatics and Engineering,
The University of Electro-Communications, Tokyo, Japan

Abstract

The minimum sum coloring problem with bundles was introduced by Darbouy and Friggstad (SWAT 2024) as a common generalization of the minimum coloring problem and the minimum sum coloring problem. During their presentation, the following open problem was raised: whether the minimum sum coloring problem with bundles could be solved in polynomial time for trees. We answer their question in the negative by proving that the minimum sum coloring problem with bundles is NP-hard even for paths. We complement this hardness by providing algorithms of the following types. First, we provide a fixed-parameter algorithm for trees when the number of bundles is a parameter; this can be extended to graphs of bounded treewidth. Second, we provide a polynomial-time algorithm for trees when bundles form a partition of the vertex set and the difference between the number of vertices and the number of bundles is constant. Third, we provide a polynomial-time algorithm for trees when bundles form a partition of the vertex set and each bundle induces a connected subgraph. We further show that for bipartite graphs, the problem with weights is NP-hard even when the number of bundles is at least three, but is polynomial-time solvable when the number of bundles is at most two. The threshold shifts to three versus four for the problem without weights.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Problems, reductions and completeness

Keywords and phrases graph algorithms, minimum sum coloring, minimum coloring, fixed-parameter tractability, NP-hardness

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.40

Related Version *Full Version*: https://doi.org/10.48550/arXiv.2509.15080

Funding *Takehiro Ito*: JSPS KAKENHI Grant Numbers JP24H00686, JP24H00690.
Naonori Kakimura: JSPS KAKENHI Grant Numbers JP21H03397, JP23K21646, JP22H05001, and JST ERATO Grant Number JPMJER2301, Japan.
Naoyuki Kamiyama: JSPS KAKENHI Grant Number JP24K14825.
Yusuke Kobayashi: JSPS KAKENHI Grant Numbers JP22H05001, JP24K02901.
Yoshio Okamoto: JSPS KAKENHI Grant Number JP23K10982.

Acknowledgements We are grateful to anonymous reviewers for their helpful comments.

© Takehiro Ito, Naonori Kakimura, Naoyuki Kamiyama, Yusuke Kobayashi, and Yoshio Okamoto;
licensed under Creative Commons License CC-BY 4.0

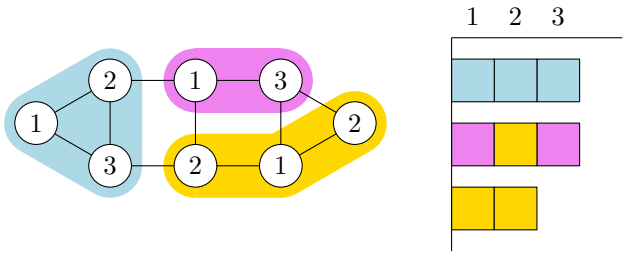


Figure 1 (Left) An instance of the minimum sum coloring problem with bundles. Bundles are distinguished by color. In this example, bundles are disjoint, but they do not have to be disjoint in general. Numbers on vertices represent the colors assigned to them. The maximum color is 3 for the violet bundle, 3 for the blue bundle, and 2 for the orange bundle; their sum is 8. (Right) Scheduling with conflicts. This is a Gantt chart, where each unit-time job corresponds to a square and the numbers correspond to time slots. Colors show bundles. The graph on the left is a conflict graph over the jobs, and the minimum sum coloring problem with bundles corresponds to minimizing the sum of makespans over bundles.

1 Introduction

The minimum sum coloring problem with bundles was introduced by Darbouy and Friggstad [10] as a common generalization of the minimum coloring problem and the minimum sum coloring problem. For the minimum sum coloring problem with bundles, we are given an undirected graph G and a family of vertex subsets of G, called bundles. Then, we want to find a (proper) coloring of G by positive integers such that the sum of the maximum colors in bundles is minimized (a precise definition will be given in the next section). Figure 1 (Left) shows an example. The minimum coloring problem corresponds to the case where the vertex set itself is a unique bundle; the minimum sum coloring problem corresponds to the case where each vertex forms a singleton bundle.

The minimum sum coloring problem with bundles can be seen as a combinatorial model for the following scheduling problem. We want to process n jobs with unit processing time. Among them, some pairs of jobs cannot be processed concurrently due to conflicts such as competing resources or precedence constraints. Those conflicts are modeled by an undirected graph in which the vertices represent given jobs, and two vertices are joined by an edge if and only if the corresponding jobs are in conflict. Furthermore, there are ℓ agents, and each of them is interested in the completion of some set of jobs. Each of these sets of jobs is modeled as a bundle, and the agents want to minimize the completion time of the last job in each of their bundles. We assume that there are sufficiently many machines with identical power. Then, an optimal solution to the minimum sum coloring problem with bundles corresponds to a scheduling of jobs that minimizes the total (or average) completion time of the last jobs in the bundles. Figure 1 (Right) shows the correspondence. With this application in mind, it is natural to assume that the bundles form a partition of the vertex set. However, some of our results can be applied even when the bundles do not form a partition.

The minimum sum coloring problem with bundles is a computationally hard problem. This is expected since the minimum coloring problem is already NP-hard [16]. However, it is known that the minimum coloring problem and the minimum sum coloring problem can be solved in linear time for trees [17]. Darbouy and Friggstad [10], in their presentation at SWAT 2024, asked whether the minimum sum coloring problem with bundles can be solved in polynomial time for trees.

The first main contribution of this paper is to prove that the minimum sum coloring problem with bundles is NP-hard even when the graph is a path and bundles form a partition of the vertex set. This answers the open question by Darbouy and Friggstad [10] mentioned above since paths are trees. On the other hand, we already know from the literature that, if the bundles form a partition, the problem is linear-time solvable when the number of bundles is one (corresponding to the minimum coloring problem) or when it is equal to the number of vertices (corresponding to the minimum sum coloring problem). We also observe that in our reduction, the subgraph induced by a bundle can be disconnected. Therefore, we wonder what happens if the number of bundles is small or large, or if each bundle induces a connected subgraph.

The second contribution of this paper is to provide a fixed-parameter algorithm for the minimum sum coloring problem with bundles in trees when the number of bundles is a parameter. The algorithm also works when each bundle is associated with a non-negative weight that acts as a coefficient in the sum. The algorithm is based on a combinatorial lemma that states that the number of colors in an optimal coloring is bounded by the chromatic number of the graph multiplied by the number of bundles, which holds for general graphs.

The third contribution is to provide a polynomial-time algorithm for the minimum sum coloring problem with bundles in trees when bundles form a partition and the difference between the number of vertices and the number of bundles is small. It should be noted that this is not a fixed-parameter algorithm with respect to that difference.

The fourth contribution is to provide a polynomial-time algorithm for the minimum sum coloring problem with bundles in trees when the family of bundles forms a partition of the vertex set and each bundle induces a connected subgraph. Note that this algorithm does not require the number of bundles to be bounded. We also show that the minimum sum coloring problem with bundles in paths can be solved in polynomial time when each bundle induces a connected subgraph (even if the bundle family does not necessarily form a partition).

Next, we turn our attention to bipartite graphs. Since for bipartite graphs, the minimum coloring (i.e., when the number of bundles is one) is easy and the minimum sum coloring (i.e., when the number of bundles is equal to the number of vertices) is hard [3], we investigate how the number of bundles affects the computational complexity of the minimum sum coloring problem with bundles.

As the fifth contribution, we prove that for bipartite graphs, the minimum sum coloring problem with bundles can be solved in polynomial time when the number of bundles is at most three, and it is NP-hard when the number of bundles is at least four. For the weighted case, the problem can be solved in polynomial time when the number of bundles is at most two, and it is NP-hard when the number of bundles is at least three.

Our results are summarized in Table 1.

Related work

As mentioned in the introduction, the minimum sum coloring problem with bundles was introduced by Darbouy and Friggstad [10] as a common generalization of the minimum coloring problem and the minimum sum coloring problem.

The minimum coloring problem has been one of the central topics in graph theory, graph algorithms, combinatorial optimization, and computational complexity theory. It is known that the minimum coloring problem is NP-complete [16],[1] even for 4-regular planar graphs [9].

[1] Whenever we talk about NP-completeness of a minimization problem, we consider a canonical decision

Table 1 Summary of the results in this paper. Here, n is the number of vertices.

	# bundles	Type of bundles	Weights	Results	Reference
Trees	Unbounded	Independent partition	Uniform	NP-complete	Theorem 3
	Parameter	General	General	FPT	Theorem 5
	n − Parameter	Partition	General	XP	Theorem 7
	Unbounded	Connected partition	General	P	Theorem 8
Bipartite	≤ 3	General	Uniform	P	Theorem 13
	≥ 4	General	Uniform	NP-complete	Theorem 10
	≤ 2	General	General	P	Theorem 12
	≥ 3	General	General	NP-complete	Theorem 11

For several classes of graphs, the minimum coloring problem can be solved in polynomial time. Most notably, for perfect graphs, a polynomial-time algorithm is known that is based on semidefinite programming relaxation [13]. The class of perfect graphs contains bipartite graphs, interval graphs, and chordal graphs, for which the minimum coloring problem can be solved in linear time.

The minimum sum coloring problem was introduced by Kubicka and Schwenk [17], where the NP-completeness of the problem was established. The problem is also known to be NP-complete even for bipartite graphs [3] and interval graphs [19, 18] (and hence for chordal graphs). We recommend a survey by Halldórsson and Kortsarz [14] on approximability for the minimum sum coloring.

The minimum coloring problem and the minimum sum coloring problem are closely related to scheduling with conflict or mutual exclusion scheduling. In a basic version of scheduling with conflict, we are given a set of n jobs with unit processing time, a set of m machines, and a conflict graph G on the set of jobs. Then, we need to assign each job to one of the machines and process the jobs along the timeline in such a way that two jobs are not processed at the same time slot if they are adjacent in G. The objective is to minimize either the makespan or the total completion time. The minimum coloring problem corresponds to the case where $m \geq n$ and the objective is makespan minimization; the minimum sum coloring problem corresponds to the case where $m \geq n$ and the objective is total completion time minimization. Baker and Coffman Jr. [2] proved that the problem of scheduling with conflict can be solved in polynomial time when $m = 2$ and is NP-hard when $m \geq 3$. For more recent and related results, see a paper by Even et al. [11]. We note that there are versions of scheduling with conflicts where we cannot process two jobs in conflict on the same machine [5].

2 Preliminaries

For a positive integer $k \in \mathbb{Z}_{>0}$, we use the notation $[k] = \{1, 2, \ldots, k\}$.

All graphs in this paper are finite, undirected, and simple (i.e., having no self-loops or multiple edges). A graph $G = (V, E)$ is *bipartite* if the vertex set V can be partitioned into two disjoint sets A and B in such a way that every edge $e \in E$ has one endpoint in A and the other endpoint in B. A graph G is a *tree* if it is connected and has no cycles. For $v \in V$, let $\delta_G(v)$ denote the set of all edges incident to v.

version of the problem in which we are also given an upper bound C of the optimal value, and want to decide whether the optimal value is at most C.

Let $G = (V, E)$ be a graph. A (proper) *coloring* of G is a map $c\colon V \to \mathbb{Z}_{>0}$ such that $c(u) \neq c(v)$ for every edge $uv \in E$. For a vertex subset $S \subseteq V$, an *S-partial coloring* of G is a map $c\colon V \to \mathbb{Z}_{\geq 0}$ such that $c(v) > 0$ for $v \in S$, $c(v) = 0$ for $v \in V \setminus S$, and $c(u) \neq c(v)$ for every edge $uv \in E$ with $u, v \in S$.

The problem Minimum Sum Coloring with Bundles is defined as follows.

Problem: Minimum Sum Coloring with Bundles
Input: An undirected graph $G = (V, E)$, a family $\mathcal{B} = \{B_1, B_2, \ldots, B_\ell\}$ of non-empty subsets of V, a weight function $w\colon \mathcal{B} \to \mathbb{Z}_{>0}$, and a positive integer $C \in \mathbb{Z}_{>0}$.
Question: Determine whether there exists a (proper) coloring $c\colon V \to \mathbb{Z}_{>0}$ such that

$$\mathrm{cost}(c) := \sum_{j=1}^{\ell} w(B_j) \max\{c(v) \mid v \in B_j\} \leq C.$$

For an instance $I = (G, \mathcal{B}, w, C)$ of Minimum Sum Coloring with Bundles, we call a coloring c of G *optimal* if $\mathrm{cost}(c) \leq \mathrm{cost}(c')$ for all colorings c' of G.

Each member of $\mathcal{B}$ is called a *bundle*. Throughout the paper, we assume that each vertex $v \in V$ is contained in at least one bundle; otherwise, we can remove v from the instance. The family $\mathcal{B}$ is a *partition* if for each vertex $v \in V$ there exists a unique bundle B_j such that $v \in B_j$. A bundle B is *connected* if the induced subgraph $G[B]$ is connected. A bundle B is *independent* if the induced subgraph $G[B]$ has no edges. The family $\mathcal{B}$ is called *connected* (and *independent*) if all its members are connected (and independent, respectively).

The following observation will be used later explicitly or implicitly.

▶ **Observation 1.** *For any graph $G = (V, E)$ in Minimum Sum Coloring with Bundles, there exists an optimal coloring in which the color of every vertex v is at most $|\delta_G(v)| + 1$.*

Proof. If the color of v exceeds $|\delta_G(v)| + 1$, then one of the colors in $\{1, 2, \ldots, |\delta_G(v)| + 1\}$ is missing in the neighborhood of v, and v can be recolored with that missing color without increasing the cost. ◀

3 Trees

3.1 NP-Completeness for Paths

As an intermediate step, we first prove that Minimum Sum Coloring with Bundles is NP-complete for perfect matchings. Here, a *perfect matching* is a set of edges in which no pair of edges shares a common endpoint.

▶ **Theorem 2.** *Minimum Sum Coloring with Bundles is NP-complete even when the edge set of the input graph forms a perfect matching, $\mathcal{B}$ is an independent partition of the vertex set, and $w(B) = 1$ for every $B \in \mathcal{B}$.*

Proof. It is easy to see that Minimum Sum Coloring with Bundles is in NP. To prove NP-hardness, we reduce Independent Set to Minimum Sum Coloring with Bundles. Here, for an undirected graph $G = (V, E)$, we say that $S \subseteq V$ is *independent* if G has no edge connecting the vertices in S.

Problem: Independent Set
Input: An undirected graph $G = (V, E)$ and a positive integer $k \in \mathbb{Z}_{>0}$.
Question: Determine whether there exists an independent set $S \subseteq V$ with $|S| \geq k$.
It is well-known that Independent Set is NP-complete (see [16]).

Suppose that we are given an instance of Independent Set that consists of a graph $G = (V, E)$ and a positive integer k. We construct a new graph $G' = (V', E')$ by replacing each vertex $v \in V$ with $|\delta_G(v)|$ new vertices so that each new vertex has exactly one incident edge. Formally, G' is defined as follows:

$$V' = \{p_{(v,e)} \mid v \in V,\ e \in \delta_G(v)\}, \qquad E' = \{p_{(u,e)}p_{(v,e)} \mid e = uv \in E\}.$$

Let $B_v := \{p_{(v,e)} \mid e \in \delta_G(v)\}$ for each $v \in V$, and define $\mathcal{B} = \{B_v \mid v \in V\}$. Then, it is easy to see that E' forms a perfect matching in G' and $\mathcal{B}$ is a partition of V'. Let $w(B) = 1$ for any $B \in \mathcal{B}$, and let $C = 2|V| - k$. This defines an instance $(G', \mathcal{B}, w, C)$ of Minimum Sum Coloring with Bundles. See Figure 2.

To show the validity of this reduction, it suffices to show that G has an independent set of size at least k if and only if there exists a coloring c of G' such that $\sum_{v \in V} \max\{c(p) \mid p \in B_v\} \leq C$.

We first show the sufficiency ("if" part). Suppose that G' has a coloring $c\colon V' \to \mathbb{Z}_{>0}$ such that $\mathrm{cost}(c) \leq C = 2|V| - k$. Let $S := \{v \in V \mid \max\{c(p) \mid p \in B_v\} = 1\}$. Then, since

$$C \geq \mathrm{cost}(c) = \sum_{v \in V} \max\{c(p) \mid p \in B_v\} \geq 2(|V| - |S|) + |S|,$$

we obtain $|S| \geq k$. For any edge $e = uv \in E$, since $c(p_{(u,e)}) \neq c(p_{(v,e)})$ as c is a coloring, at least one of $c(p_{(u,e)}) \geq 2$ and $c(p_{(v,e)}) \geq 2$ holds, which implies that at least one of u and v is in $V \setminus S$. Therefore, S is an independent set in G, which shows the sufficiency.

We next show the necessity ("only if" part). Suppose that G has an independent set $S \subseteq V$ with $|S| \geq k$. Let $c\colon V' \to \mathbb{Z}_{>0}$ be a coloring of G' such that

- for each edge $e = uv \in E$, one of $c(p_{(u,e)})$ and $c(p_{(v,e)})$ is 1 and the other is 2, and
- for any $v \in S$ and any $e \in \delta_G(v)$, it holds that $c(p_{(v,e)}) = 1$.

Note that such a coloring c exists, because S is an independent set in G. Since $\max\{c(p) \mid p \in B_v\} = 1$ for $v \in S$ and $\max\{c(p) \mid p \in B_v\} \leq 2$ for $v \in V \setminus S$, we obtain

$$\mathrm{cost}(c) = \sum_{v \in V} \max\{c(p) \mid p \in B_v\} \leq 2|V| - |S| \leq C,$$

which shows the necessity.

Therefore, the reduction is valid, and hence Minimum Sum Coloring with Bundles is NP-hard even when the edge set forms a perfect matching, $\mathcal{B}$ is an independent partition of the vertex set, and $w(B) = 1$ for every $B \in \mathcal{B}$. ◀

We then prove the NP-hardness of Minimum Sum Coloring with Bundles on paths by reducing Minimum Sum Coloring with Bundles with the conditions in Theorem 2. Roughly speaking, the reduction involves creating four copies of the instance described in Theorem 2 and carefully connecting their edges to form a single path. See the full version for details.

▶ **Theorem 3.** *Minimum Sum Coloring with Bundles is NP-complete even when G is a path, $\mathcal{B}$ is an independent partition of the vertex set, and $w(B) = 1$ for every $B \in \mathcal{B}$.*

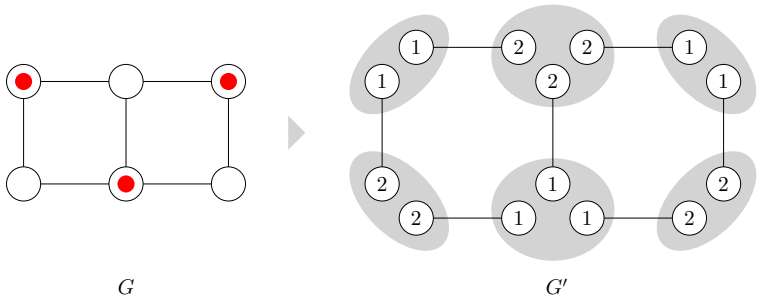


Figure 2 Reduction when a graph forms a perfect matching.

3.2 Fixed-Parameter Tractability when $|\mathcal{B}|$ is a Parameter

In the reduction of the previous section, the number of bundles is unbounded. As a positive algorithmic result that is in contrast to Theorem 3, we prove that Minimum Sum Coloring with Bundles in graphs of bounded treewidth can be solved in fixed-parameter tractable time when the number of bundles and the treewidth are parameters. See a textbook (e.g. [8]) for an introduction to treewidths and Courcelle's theorem.

First, we prove the lemma that bounds the number of colors in any optimal coloring. For a graph $G = (V, E)$, we denote by $\chi(G)$ the *chromatic number* of G, which is defined as the minimum of $\max_{v \in V} c(v)$ over all (proper) colorings c of G.

▶ **Lemma 4.** *Let $I = (G = (V, E), \mathcal{B}, w, C)$ be an instance of Minimum Sum Coloring with Bundles such that each vertex is contained in some bundle. Then, every optimal coloring $c\colon V \to \mathbb{Z}_{>0}$ for I satisfies $c(v) \leq \chi(G)|\mathcal{B}|$ for all $v \in V$.*

Proof. The proof proceeds by induction on $|\mathcal{B}|$. Consider the case where $|\mathcal{B}| = 1$ and $\mathcal{B} = \{B\}$. Note that $B = V$ as each vertex is contained in some bundle. Let c be a minimum coloring, i.e., a coloring that attains $\chi(G)$. Then, $\max_{v \in B} c(v) \leq \max_{v \in V} c(v) = \chi(G) = \chi(G)|\mathcal{B}|$.

Now, let $\ell \geq 1$ be a positive integer, and assume that the lemma holds when the number of bundles is ℓ. Then, we prove that the lemma holds when the number of bundles is $\ell + 1$. For the sake of contradiction, suppose that there exists an optimal coloring c^* such that $\max_{v \in B} c^*(v) > \chi(G)(\ell + 1)$ for some $B \in \mathcal{B}$. Fix such a member B, and let U be the set of vertices of G that belong to B, but not to other bundles. Then, consider the instance $I' = (G - U, \mathcal{B} \setminus \{B\}, w|_{V \setminus U}, C)$, where $w|_{V \setminus U}$ is the restriction of w on $V \setminus U$. By the induction hypothesis, there exists an optimal coloring $c'\colon V \setminus U \to \mathbb{Z}_{>0}$ for I' such that $\max_{v \in V \setminus U} c'(v) \leq \chi(G - U)\ell \leq \chi(G)\ell$. We now extend c' to a coloring c of G by giving colors in $\{\chi(G)\ell + 1, \ldots, \chi(G)\ell + \chi(G)\}$ to the vertices of U. This is possible since $\chi(G[U]) \leq \chi(G)$. Then,

$$\begin{aligned}
\mathrm{cost}(c) &= \sum_{B' \in \mathcal{B} \setminus \{B\}} w(B') \max_{v \in B'} c(v) + w(B) \max_{v \in B} c(v) \\
&\leq \sum_{B' \in \mathcal{B} \setminus \{B\}} w(B') \max_{v \in B'} c'(v) + w(B)\chi(G)(\ell + 1) \\
&< \sum_{B' \in \mathcal{B} \setminus \{B\}} w(B') \max_{v \in B'} c^*(v) + w(B) \max_{v \in B} c^*(v) \ = \mathrm{cost}(c^*).
\end{aligned}$$

This contradicts the optimality of c^*. ◀

We now focus on the case where the input graph G has treewidth at most tw. Note that $\chi(G) \leq \mathsf{tw}+1$, because the degeneracy of G is at most tw. Thus, the number of colors in any optimal coloring is at most $(\mathsf{tw}+1)|\mathcal{B}|$ by Lemma 4.

To solve Minimum Sum Coloring with Bundles for graphs $G=(V,E)$ with treewidth at most tw, we employ Courcelle's theorem [1, 6, 7]. To this end, we introduce the following auxiliary decision problem. In addition to the input $(G,\mathcal{B},w,C)$ to Minimum Sum Coloring with Bundles, where $\mathcal{B}=\{B_1,B_2,\ldots,B_\ell\}$, we also take a sequence $\mathbf{k}=(k_1,k_2,\ldots,k_\ell)$ of integers. Then, we want to decide whether there exists a coloring $c\colon V\to\mathbb{Z}_{>0}$ such that $\max_{v\in B_j} c(v)\leq k_j$ for all $j\in[\ell]$. If we have a solution to this auxiliary problem for every $\mathbf{k}\in[(\mathsf{tw}+1)\ell]^\ell$, then the optimal cost for the instance $(G,\mathcal{B},w,C)$ can be derived as

$$\min_{\mathbf{k}}\left\{\sum_{j=1}^{\ell} w(B_j)k_j \;\middle|\; \text{the answer to the instance } (G,\mathcal{B},w,C,\mathbf{k}) \text{ is yes}\right\}. \tag{1}$$

We now focus on solving the auxiliary problem above. To solve the auxiliary problem using Courcelle's theorem [1, 6, 7], it is sufficient to write down an MSO formula that expresses the decision. An instance is given as $(G,\mathcal{B},w,C,\mathbf{k})$, and we want to decide whether there exists a coloring c such that $\max_{v\in B_j} c(v)\leq k_j$ for all $j\in[\ell]$. By Lemma 4, we can assume that our coloring c uses colors in $[(\mathsf{tw}+1)\ell]$. For brevity, let $p=(\mathsf{tw}+1)\ell$. To express conditions with an MSO formula, we regard a coloring c as a partition $\{C_1,C_2,\ldots,C_p\}$ of the vertex set V into independent sets, where $C_i=\{v\in V\mid c(v)=i\}$. Note that in such a partition a set C_i can be empty.

Then, the decision can be expressed by the following MSO formula:

$$\exists\, C_1,C_2,\ldots,C_p\subseteq V\colon \bigwedge_{i=1}^{p}\bigwedge_{i'=i+1}^{p} \forall\, v\in V\colon \neg(v\in C_i\wedge v\in C_{i'}) \tag{2}$$

$$\wedge\, \forall\, v\in V\colon \bigvee_{i=1}^{p}(v\in C_i) \tag{3}$$

$$\wedge \bigwedge_{i=1}^{p} \forall\, u,v\in V\colon\ (u\in C_i\wedge v\in C_i\to\neg(uv\in E)) \tag{4}$$

$$\wedge \bigwedge_{j=1}^{\ell} \forall\, v\in V\colon \left(v\in B_j\to\bigvee_{i=1}^{k_j}(v\in C_i)\right). \tag{5}$$

In the formula, the conjunction of Formulae (2) and (3) represents that $\{C_1,C_2,\ldots,C_p\}$ is a partition of V. Formula (4) represents that each C_i is independent. Formula (5) represents that the maximum color in bundle B_j is at most k_j for each $j\in[\ell]$. Thus, the formula correctly represents the conditions in the auxiliary problem.

By Courcelle's theorem, this auxiliary problem can be solved in $O(f(\mathsf{tw},\ell)|V|)$ time for some computable function f. To evaluate the minimum in (1), we solve the auxiliary problem for every choice of $\mathbf{k}\in[(\mathsf{tw}+1)\ell]^\ell$, thus at most $((\mathsf{tw}+1)\ell)^\ell$ times. Let $g(\mathsf{tw})|V|$ be the running time of Bodlaender's algorithm [4] for computing a tree decomposition of width tw. Then, the overall running time is $g(\mathsf{tw})|V|+((\mathsf{tw}+1)\ell)^\ell\cdot O(f(\mathsf{tw},\ell)|V|)$. Setting $h(\mathsf{tw},\ell)=g(\mathsf{tw})+((\mathsf{tw}+1)\ell)^\ell f(\mathsf{tw},\ell)$, we obtain the following theorem.

▶ **Theorem 5.** *Minimum Sum Coloring with Bundles can be solved in $O(h(\mathsf{tw},\ell)|V|)$ time for some computable function $h\colon\mathbb{Z}_{\geq 0}\times\mathbb{Z}_{\geq 0}\to\mathbb{Z}_{\geq 0}$, where tw is the treewidth of G and ℓ is the number of bundles.* ◀

3.3 An XP-Algorithm when $|V| - |\mathcal{B}|$ is a Parameter

We consider the case where G is a tree, the bundle family is a partition of the vertex set, and the number of bundles is large. More specifically, if n is the number of vertices and ℓ is the number of bundles that form a partition of the vertex set, we treat $n - \ell$ as a parameter.

Let $B \in \mathcal{B}$ be a bundle. We call B a *singleton bundle* if $|B| = 1$; otherwise, we call B a *non-singleton bundle*.

▶ **Lemma 6.** *Let $I = (G = (V, E), \mathcal{B}, w, C)$ be an instance of* Minimum Sum Coloring with Bundles, *where $\mathcal{B}$ is a partition of V. Then, the number of non-singleton bundles in $\mathcal{B}$ is at most $|V| - |\mathcal{B}|$.*

Proof. Let t be the number of non-singleton bundles. Since $\mathcal{B}$ is a partition of V, we obtain $|V| \geq 2t + (|\mathcal{B}| - t) = |\mathcal{B}| + t$. Thus, $t \leq |V| - |\mathcal{B}|$. ◀

Let $I = (G = (V, E), \mathcal{B}, w, C)$ be an instance of Minimum Sum Coloring with Bundles, and assume that G is a tree, $\mathcal{B}$ is a partition of V. Let t be the number of non-singleton bundles in $\mathcal{B}$. By Lemma 6, $t \leq |V| - |\mathcal{B}| = n - \ell$. Denote the non-singleton bundles in $\mathcal{B}$ by $B_1, B_2, \ldots, B_t$, and the set of vertices, each of which forms a singleton bundle, by V_s.

A primary idea of our algorithm is to fix upper bounds on the colors used in non-singleton bundles and to optimize the colors in singleton bundles with this restriction. For a non-singleton bundle B_j, $j \in [t]$, let k_j be an upper bound on the colors in B_j; namely, we seek a coloring c such that $\max_{u \in B_j} c(u) \leq k_j$ for every $j \in [t]$. For brevity, we denote $\mathbf{k} = (k_1, k_2, \ldots, k_t)$. By Lemma 4, it is enough to consider the situations with $k_j \leq 2\ell$ for all $j \in [t]$, and thus the number of possible choices for $\mathbf{k}$ is bounded by $(2\ell)^t$ from above.

We regard G as a rooted tree with a root $r \in V$. For each vertex $v \in V$, we denote by $V(v)$ the vertex set of the subtree of G rooted at v. For each vertex $v \in V$, each color k, and an upper-bound tuple $\mathbf{k}$, we define

$$f(v, k; \mathbf{k}) := \min \left\{ \sum_{u \in V_s} w(\{u\}) c(u) \;\middle|\; \begin{array}{l} c \text{ is a } V(v)\text{-partial coloring of } G, \\ c(v) = k, \\ \max_{u \in B_j} c(u) \leq k_j \text{ for all } j \in [t] \end{array} \right\}.$$

Then, the optimal value for the instance I is read by

$$\min_{\mathbf{k}, k} \left(f(r, k; \mathbf{k}) + \sum_{j=1}^{t} k_j \right),$$

where the minimum is taken over all possible choices of $\mathbf{k} = (k_1, k_2, \ldots, k_t)$ and k. Note that we only need to consider the value of k in the interval $k \in [2\ell]$ by Lemma 4.

We now establish a recursive formula for $f(v, k; \mathbf{k})$. First, consider the case where v is a leaf of G. We have two subcases. If v forms a singleton bundle by itself, then

$$f(v, k; \mathbf{k}) = w(\{v\})k.$$

If v belongs to a non-singleton bundle B_j for some $j \in [t]$, which is unique since $\mathcal{B}$ forms a partition of V, then

$$f(v, k; \mathbf{k}) = \begin{cases} 0 & \text{if } k \leq k_j, \\ +\infty & \text{otherwise.} \end{cases}$$

Next, consider the case where v is not a leaf of G. Let $v_1, \ldots, v_z$ be the children of v in G. We again have two subcases. If v forms a singleton bundle by itself, then

$$f(v,k;\mathbf{k}) = w(\{v\})k + \min\left\{ \sum_{y=1}^{z} f(v_y, k'_y; \mathbf{k}) \,\middle|\, k'_y \neq k \text{ for all } y \in [z] \right\}$$
$$= w(\{v\})k + \sum_{y=1}^{z} \min\left\{ f(v_y, k'_y; \mathbf{k}) \,\middle|\, k'_y \neq k \right\}$$

since the values $f(v_y, k'_y; \mathbf{k})$ are independent from each other. Similarly, if v belongs to a non-singleton bundle B_j for some $j \in [t]$, then

$$f(v,k;\mathbf{k}) = \begin{cases} \sum_{y=1}^{z} \min\left\{ f(v_y, k'_y; \mathbf{k}) \,\middle|\, k'_y \neq k \right\} & \text{if } k \leq k_j, \\ +\infty & \text{otherwise.} \end{cases}$$

For each fixed $\mathbf{k}$, we compute $f(v,k;\mathbf{k})$ in a bottom-up manner from leaves to root. When v is not a leaf, the computation of $f(v,k;\mathbf{k})$ can be done in $O(z\ell)$ time for each fixed k since we may suppose that $k'_y \leq 2\ell$ by Lemma 4. In total, for each fixed $\mathbf{k}$, the computation of $f(v,k;\mathbf{k})$ over all v and k can be done in $O(n\ell^2)$ time. Hence, for varying $\mathbf{k}$, the running time of our algorithm is $O((2\ell)^t n\ell^2)$. When $n - \ell$ is a parameter, this is an XP algorithm since $t \leq n - \ell$ by Lemma 6.

We summarize our findings in the following theorem.

▶ **Theorem 7.** *Minimum Sum Coloring with Bundles can be solved in $O((2\ell)^t n\ell^2)$ time when G is a tree, n is the number of vertices, ℓ is the number of bundles, and t is the number of non-singleton bundles.*

3.4 Polynomial-time Algorithm for the Connected Partition Case

In this subsection, we consider Minimum Sum Coloring with Bundles in trees when the bundle family is a connected partition, namely, when each bundle induces a connected subgraph and each vertex belongs to a unique bundle.

Let $I = (G = (V,E), \mathcal{B}, w, C)$ be an instance of Minimum Sum Coloring with Bundles, and assume that G is a tree, $\mathcal{B}$ is a connected partition of V. Define $n := |V|$. For each vertex $v \in V$, we denote by $B(v)$ a unique bundle in $\mathcal{B}$ containing v. Notice that there exists an optimal coloring c such that $c(v) \in [n]$ for every vertex $v \in V$ by Observation 1.

We regard G as a rooted tree with a root $r \in V$. For each vertex $v \in V$, we denote by $V(v)$ the vertex set of the subtree of G rooted at v. For each vertex $v \in V$, each integer $k \in [n]$, and each integer $g \in [n]$, we define

$$f(v,k,g) := \min\left\{ \sum_{B \in \mathcal{B}} w(B) \max_{u \in B} c(u) \,\middle|\, \begin{array}{l} c \text{ is a } V(v)\text{-partial coloring of } G, \\ c(u) \leq n \text{ for all } u \in V, \\ c(v) = k, \\ \max\{c(u) \mid u \in B(v)\} = g \end{array} \right\}.$$

We now establish a recursive formula for $f(v,k,g)$. First, consider the case where v is a leaf of G. For $k \in [n]$ and $g \in [n]$, we obtain

$$f(v,k,g) = \begin{cases} w(B(v))g & \text{if } k = g, \\ +\infty & \text{otherwise.} \end{cases}$$

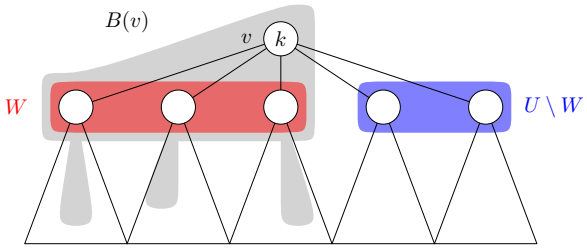


Figure 3 Notation for the case where $\mathcal{B}$ is a partition of V.

Next, consider the case where v is not a leaf of G. Let U be the set of children of v and let $W := U \cap B(v)$. See Figure 3. Notice that $B(u) = B(v)$ for every $u \in W$. Furthermore, since $B(v)$ is connected, $B(v) \cap V(u) = \emptyset$ for every $u \in U \setminus W$. If $k > g$, then we obtain $f(v,k,g) = +\infty$ since $c(v) \leq \max\{c(u) \mid u \in B(v)\}$. Since $\mathcal{B}$ is a connected partition of V, $|\{u \in U \mid B \cap V(u) \neq \emptyset\}| \leq 1$ for each bundle $B \in \mathcal{B} \setminus \{B(v)\}$. Thus, if $k = g$, then

$$f(v,k,g) = w(B(v))g + \sum_{u \in W} \min_{k' \in [n]\setminus\{k\}, g' \in [g]} \Big(f(u,k',g') - w(B(u))g' \Big) + \sum_{u \in U \setminus W} \min_{k' \in [n]\setminus\{k\}, g' \in [n]} f(u,k',g').$$

Similarly, if $k < g$, then

$$f(v,k,g) = \min_{u' \in W} \Big\{ \min_{k' \in [n]\setminus\{k\}} f(u',k',g) + \sum_{u \in W \setminus \{u'\}} \min_{k' \in [n]\setminus\{k\}, g' \in [g]} \Big(f(u,k',g') - w(B(u))g' \Big) + \sum_{x \in U \setminus W} \min_{k' \in [n]\setminus\{k\}, g' \in [n]} f(u,k',g') \Big\}.$$

By computing $f(r,k,g)$ over all k and g with these equations, we can solve Minimum Sum Coloring with Bundles in this case. For each vertex $v \in V$, we can compute $f(v,k,g)$ over all k and g in $O(|\delta_G(v)|^2 n^4)$ time. Thus, we obtain the following theorem.

▶ **Theorem 8.** *Minimum Sum Coloring with Bundles can be solved in $O(n^6)$ time when G is a tree, the subgraph of G induced by each bundle $B \in \mathcal{B}$ is connected, and $\mathcal{B}$ is a partition of V.*

Finally, we present the following theorem, asserting that if G is a path, we can solve in polynomial time Minimum Sum Coloring with Bundles when $\mathcal{B}$ is connected but not necessarily a partition. See the full version for the proof.

▶ **Theorem 9.** *Minimum Sum Coloring with Bundles can be solved in $O(|V|^3|\mathcal{B}|)$ time when $G = (V,E)$ is a path and the subgraph of G induced by each bundle $B \in \mathcal{B}$ is connected.*

4 Bipartite Graphs

In this section, we first prove that Minimum Sum Coloring with Bundles is NP-complete even when G is a bipartite graph, the number of bundles is four, and the weights are uniform. To prove NP-completeness, we use the following problem.

Problem: Bipartite 3-List-Coloring
Input: A bipartite graph $G = (V, E)$ and a list $L(v) \subseteq \{1, 2, 3\}$ of available colors for each vertex $v \in V$.
Question: Determine whether there exists a proper coloring $c\colon V \to \{1, 2, 3\}$ such that $c(v) \in L(v)$ for each $v \in V$.

As observed by Golovach and Paulusma [12], a reduction given by Jansen and Scheffler [15] proves the NP-completeness of Bipartite 3-List-Coloring.

▶ **Theorem 10.** *Minimum Sum Coloring with Bundles is NP-complete even when G is a bipartite graph, $|\mathcal{B}| = 4$, and $w(B) = 1$ for every $B \in \mathcal{B}$.*

Proof. It is easy to see that the problem is in NP. In what follows, we prove that the problem is NP-hard by reducing Bipartite 3-List-Coloring.

Suppose that we are given an instance of the problem Bipartite 3-List-Coloring. Specifically, let $G = (V, E)$ be a bipartite graph, and let $L(v) \subseteq \{1, 2, 3\}$ be a list of available colors for each vertex $v \in V$. For $Z \subseteq \{1, 2, 3\}$, we define $U_Z = \{v \in V \mid L(v) = Z\}$. Then $\{U_Z \mid Z \subseteq \{1, 2, 3\}\}$ forms a partition of V; that is, $V = \bigcup_{Z \subseteq \{1,2,3\}} U_Z$ and $U_Z \cap U_{Z'} = \emptyset$ for any distinct subsets $Z, Z' \subseteq \{1, 2, 3\}$.

We construct an instance $(\tilde{G}, \mathcal{B}, w, C)$ of Minimum Sum Coloring with Bundles, where $\tilde{G} = (\tilde{V}, \tilde{E})$, as follows. For $Z \in \{\{2\}, \{2, 3\}\}$, we denote by U_Z^1 a copy of U_Z, where $u^1 \in U_Z^1$ means a copy of $u \in U_Z$. Similarly, we define two copies $U_{\{1,3\}}^1$ and $U_{\{1,3\}}^2$ of $U_{\{1,3\}}$, and three copies $U_{\{3\}}^1$, $U_{\{3\}}^2$, and $U_{\{3\}}^3$ of $U_{\{3\}}$. Then, the vertex set $\tilde{V}$ of $\tilde{G}$ is defined as

$$\tilde{V} = V \cup U_{\{2\}}^1 \cup \bigcup_{i=1}^{3} U_{\{3\}}^i \cup \bigcup_{i=1}^{2} U_{\{1,3\}}^i \cup U_{\{2,3\}}^1 \cup \{v_i \mid i \in [16]\}.$$

The edge set $\tilde{E}$ is defined as

$$\tilde{E} = E \cup E_{\{2\}} \cup E_{\{3\}} \cup E_{\{1,3\}} \cup E_{\{2,3\}} \cup E_0,$$

where we define

$$\begin{aligned} E_{\{2\}} &= \{uu^1 \mid u \in U_{\{2\}}\}, & E_{\{3\}} &= \{uu^2, u^2u^1, uu^3 \mid u \in U_{\{3\}}\}, \\ E_{\{1,3\}} &= \{uu^2, u^2u^1 \mid u \in U_{\{1,3\}}\}, & E_{\{2,3\}} &= \{uu^1 \mid u \in U_{\{2,3\}}\}, \\ E_0 &= \{v_1v_2, v_{11}v_{12}\} \cup \{v_iv_{i+1} \mid i \in \{3, 4, 5\} \cup \{7, 8, 9\} \cup \{13, 14, 15\}\}. \end{aligned}$$

We note that each of $E_{\{2\}}$, $E_{\{3\}}$, $E_{\{1,3\}}$, and $E_{\{2,3\}}$ consists of disjoint edges and paths. Also, E_0 contains 3 disjoint paths of length 3 and 2 disjoint edges. Since $\tilde{G}$ is obtained from the bipartite graph G by adding edges of $E_{\{2\}} \cup E_{\{3\}} \cup E_{\{1,3\}} \cup E_{\{2,3\}} \cup E_0$, the constructed graph $\tilde{G}$ is bipartite. Moreover, define a family of bundles $\mathcal{B} = \{B_1, B_2, B_3, B_4\}$ as

$$\begin{aligned} B_1 &= U_{\{1\}} \cup U_{\{2\}}^1 \cup U_{\{3\}}^1 \cup U_{\{3\}}^3 \cup U_{\{1,3\}}^1 \cup U_{\{2,3\}}^1 \cup \{v_1, v_3, v_6\}, \\ B_2 &= U_{\{2\}} \cup U_{\{1,2\}} \cup U_{\{3\}}^2 \cup U_{\{1,3\}}^2 \cup \{v_2, v_7, v_{10}, v_{12}\}, \\ B_3 &= U_{\{3\}} \cup U_{\{1,3\}} \cup U_{\{2,3\}} \cup U_{\{1,2,3\}} \cup \{v_4, v_5, v_8, v_9, v_{14}, v_{15}\}, \\ B_4 &= \{v_{11}, v_{13}, v_{16}\}. \end{aligned}$$

We set $w(B_i) = 1$ for $i \in \{1, 2, 3, 4\}$ and $C = 7$. This defines an instance $(\tilde{G}, \mathcal{B}, w, C)$ of Minimum Sum Coloring with Bundles. See Figure 4.

We claim that the instance $(\tilde{G}, \mathcal{B}, w, C)$ of Minimum Sum Coloring with Bundles is a yes-instance if and only if the given instance (G, L) of Bipartite 3-List-Coloring is a yes-instance. The proof of this claim can be found in the full version. ◀

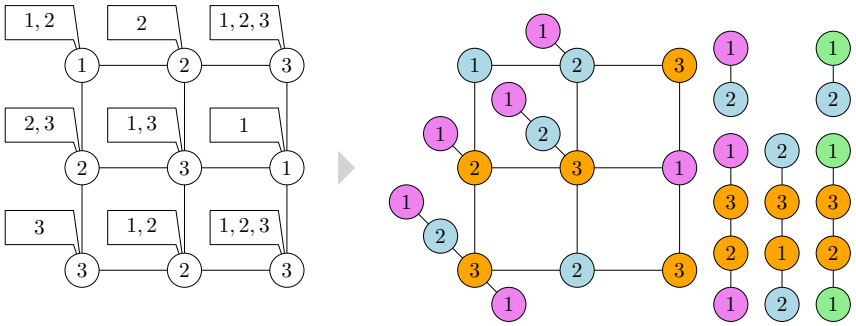


Figure 4 Reduction for bipartite graphs with uniform weight. (Left) An instance of Bipartite 3-List-Coloring. The flag attached to each vertex v represents the list $L(v)$. The numbers on vertices represent a proper coloring that respects the lists. (Right) The instance of Minimum Sum Coloring with Bundles that is constructed from the left instance of Bipartite 3-List-Coloring. Colors represent bundles: the violet vertices form B_1, the blue vertices form B_2, the orange vertices form B_3, and the green vertices form B_4. The numbers on vertices represent the coloring obtained as in the proof of the if-part.

For the weighted case, Minimum Sum Coloring with Bundles on bipartite graphs is NP-complete even when $|\mathcal{B}| = 3$. The proof is an adaptation of that for Theorem 10, and can be found in the full version.

▶ **Theorem 11.** *Minimum Sum Coloring with Bundles is NP-complete even when G is a bipartite graph, $|\mathcal{B}| = 3$, and $w(B) \in \{1, 2\}$ for every $B \in \mathcal{B}$.*

To complement Theorems 10 and 11 concerning bipartite graphs, we prove that Minimum Sum Coloring with Bundles can be solved in polynomial time under two specific conditions: when $|\mathcal{B}| \leq 2$ and when $|\mathcal{B}| \leq 3$ with uniform weights. The proofs can be found in the full version.

▶ **Theorem 12.** *Minimum Sum Coloring with Bundles can be solved in polynomial time when G is bipartite and $|\mathcal{B}| \leq 2$.*

▶ **Theorem 13.** *Minimum Sum Coloring with Bundles with uniform weight can be solved in polynomial time when G is bipartite and $|\mathcal{B}| \leq 3$.*

5 Concluding Remarks

We have completed a detailed complexity analysis of Minimum Sum Coloring with Bundles for trees and bipartite graphs, and solved an open problem by Darbouy and Friggstad [10]. Here, we pose a few open problems. First, we do not know the existence of an FPT algorithm for trees when $|V| - |\mathcal{B}|$ is a parameter and the bundles form a partition of V. Second, we do not know the complexity for bipartite graphs when the bundles form an independent partition or a connected partition. Third, we do not know any better approximation ratio for trees and bipartite graphs than the one proposed by Darbouy and Friggstad [10].

References

1 Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12(2):308–340, 1991. doi:10.1016/0196-6774(91)90006-K.

2 Brenda S. Baker and Edward G. Coffman Jr. Mutual exclusion scheduling. *Theor. Comput. Sci.*, 162(2):225–243, 1996. doi:10.1016/0304-3975(96)00031-X.

3 Amotz Bar-Noy and Guy Kortsarz. Minimum color sum of bipartite graphs. *J. Algorithms*, 28(2):339–365, 1998. doi:10.1006/JAGM.1998.0938.

4 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.

5 Hans L. Bodlaender, Klaus Jansen, and Gerhard J. Woeginger. Scheduling with incompatible jobs. *Discret. Appl. Math.*, 55(3):219–232, 1994. doi:10.1016/0166-218X(94)90009-4.

6 Bruno Courcelle. The monadic second-order logic of graphs I: recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

7 Bruno Courcelle. The monadic second-order logic of graphs III: tree-decompositions, minor and complexity issues. *RAIRO Theor. Informatics Appl.*, 26:257–286, 1992. doi:10.1051/ITA/1992260302571.

8 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

9 David P. Dailey. Uniqueness of colorability and colorability of planar 4-regular graphs are NP-complete. *Discret. Math.*, 30(3):289–293, 1980. doi:10.1016/0012-365X(80)90236-8.

10 Seyed Parsa Darbouy and Zachary Friggstad. Approximating minimum sum coloring with bundles. In Hans L. Bodlaender, editor, *19th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2024, June 12-14, 2024, Helsinki, Finland*, volume 294 of *LIPIcs*, pages 21:1–21:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.SWAT.2024.21.

11 Guy Even, Magnús M. Halldórsson, Lotem Kaplan, and Dana Ron. Scheduling with conflicts: online and offline algorithms. *J. Sched.*, 12(2):199–224, 2009. doi:10.1007/S10951-008-0089-1.

12 Petr A. Golovach and Daniël Paulusma. List coloring in the absence of two subgraphs. *Discret. Appl. Math.*, 166:123–130, 2014. doi:10.1016/J.DAM.2013.10.010.

13 Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization, Second Edition*. Springer Belin Heidelberg, 1993. doi:10.1007/978-3-642-78240-4.

14 Magnús M. Halldórsson and Guy Kortsarz. Algorithms for chromatic sums, multicoloring, and scheduling dependent jobs. In Teofilo F. Gonzalez, editor, *Handbook of Approximation Algorithms and Metaheuristics, Second Edition, Volume 1: Methologies and Traditional Applications*, pages 671–684. Chapman and Hall/CRC, 2018. doi:10.1201/9781351236423-38.

15 Klaus Jansen and Petra Scheffler. Generalized coloring for tree-like graphs. *Discret. Appl. Math.*, 75(2):135–155, 1997. doi:10.1016/S0166-218X(96)00085-6.

16 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_9.

17 Ewa M. Kubicka and Allen J. Schwenk. An introduction to chromatic sums. In Arthur M. Riehl, editor, *Computer Trends in the 1990s - Proceedings of the 1989 ACM 17th Annual Computer Science Conference, Louisville, Kentucky, USA, February 21-23, 1989*, pages 39–45. ACM, 1989. doi:10.1145/75427.75430.

18 Dániel Marx. A short proof of the NP-completeness of minimum sum interval coloring. *Oper. Res. Lett.*, 33(4):382–384, 2005. doi:10.1016/J.ORL.2004.07.006.

19 Tibor Szkaliczki. Routing with minimum wire length in the dogleg-free Manhattan model is NP-complete. *SIAM J. Comput.*, 29(1):274–287, 1999. doi:10.1137/S0097539796303123.

A General Framework for Finding Diverse Solutions via Network Flow and Its Applications

Yuni Iwamasa
Graduate School of Informatics, Kyoto University, Japan

Tomoki Matsuda
School of Computing, Institute of Science Tokyo, Japan

Shunya Morihira
Graduate School of Informatics, Kyoto University, Japan

Hanna Sumita
School of Computing, Institute of Science Tokyo, Japan

Abstract

In this paper, we present a general framework for efficiently computing diverse solutions to combinatorial optimization problems. Given a problem instance, the goal is to find k solutions that maximize a specified diversity measure – the sum of pairwise Hamming distances or the size of the union of the k solutions. Our framework applies to problems satisfying two structural properties: (i) All solutions are of equal size and (ii) the family of all solutions can be represented by a surjection from the family of ideals of some finite poset. Under these conditions, we show that the problem of computing k diverse solutions can be reduced to the minimum cost flow problem and the maximum s-t flow problem. As applications, we demonstrate that both the unweighted minimum s-t cut problem and the stable matching problem satisfy the requirements of our framework. By utilizing the recent advances in network flows algorithms, we improve the previously known time complexities of the diverse problems, which were based on submodular function minimization.

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

Keywords and phrases Diverse Solutions, Network Flow Algorithm, Lattice Theory

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.41

Related Version *Full Version*: `https://arxiv.org/abs/2504.17633`

Funding *Yuni Iwamasa*: Partially supported by JSPS KAKENHI Grant Numbers JP22K17854, JP24K02901, JP24K21315, Japan.
Hanna Sumita: Partially supported by JST ERATO Grant Number JPMJER2301, JST ASPIRE Grant Number JPMJAP2302, and JSPS KAKENHI Grant Numbers JP21K17708, JP21H03397, Japan.

Acknowledgements We are grateful to Yasunori Kinoshita for his insightful suggestion on the start of this work. We also thank Yasuaki Kobayashi and Yutaro Yamaguchi for bibliographic information.

1 Introduction

In modeling real-world problems as optimization problems, many factors are inevitably omitted from the model due to the limited information and the need for tractability. Consequently, finding a single optimal solution to the optimization problem may not be enough. Motivated by this issue, developing algorithms to find *diverse* multiple solutions has recently gained attention in the field of combinatorial optimization [3, 4, 7, 8, 9, 10, 11, 15, 17].

We formally define the *k-diverse problem* for a positive integer k. Let d_{div} be a function that measures the "diversity" for a k-tuple of subsets of some arbitrary fixed ground set E. For a combinatorial problem Prob, the *k-diverse problem* of Prob asks, given an instance $\mathbf{I}$ of Prob, to find a k-tuple $(S_1, S_2, \ldots, S_k)$ of solutions of $\mathbf{I}$ that maximizes the fixed measure d_{div}. Typical examples of the measure of diversity include the following:

© Yuni Iwamasa, Tomoki Matsuda, Shunya Morihira, and Hanna Sumita;
licensed under Creative Commons License CC-BY 4.0

$$d_{\mathrm{sum}}(S_1, S_2, \ldots, S_k) \coloneqq \sum_{1 \le i < j \le k} |S_i \triangle S_j|, \qquad d_{\mathrm{cov}}(S_1, S_2, \ldots, S_k) \coloneqq \left| \bigcup_{1 \le i \le k} S_i \right|.$$

If we use d_{sum} (resp. d_{cov}) as the measure, then we refer to the corresponding k-diverse problem as Sum-k-Diverse (resp. Cov-k-Diverse).

In this paper, we present a general framework for efficiently finding diverse solutions to combinatorial optimization problems. Namely, we focus on Sum-k-Diverse/Cov-k-Diverse of a combinatorial problem Prob having two properties (S) and (R) introduced below. The first property (S) states that all solutions of each instance of Prob have the same size:

(S) For any instance $\mathbf{I}$ of Prob, there exist a finite ground set E and a positive integer q such that the family $\mathcal{S}(\mathbf{I})$ of solutions of $\mathbf{I}$ consists of subsets of E of size q.

The second property (R) indicates that $\mathcal{S}(\mathbf{I})$ admits a lattice structure and $\mathcal{S}(\mathbf{I})$ has a compact representation using the lattice. To state this property precisely, we first introduce some notation. Let $(P, \preceq)$ be a finite poset with minimum element $\bot$ and maximum element $\top$. For such a poset P, let P^* denote the poset obtained from P by removing $\bot$ and $\top$. For a map $r\colon E \to P^2$ that assigns each element $e \in E$ to an ordered pair

$$r(e) \coloneqq (e^+, e^-)$$

of elements in P with $e^+ \preceq e^-$, we define a set $\sup_r(I)$ by

$$\sup_r(I) \coloneqq \{e \in E \mid e^+ \in I \cup \{\bot\} \not\ni e^-\}$$

for each ideal $I \in \mathcal{I}(P^*)$. Here, an *ideal* of a poset is a subset $I \subseteq P^*$ that is closed downward under $\preceq$, that is, $y \preceq x \in I$ implies $y \in I$ for all $x, y \in P^*$, and $\mathcal{I}(P^*)$ denotes the family of ideals of the poset P^*. With this notation, the property (R) can be stated as follows:

(R) For any instance $\mathbf{I}$ of Prob, there exist a finite poset $(P, \preceq)$ with distinct minimum element $\bot$ and maximum element $\top$ and a map $r\colon E \to P^2$ given by $e \mapsto (e^+, e^-)$ with $e^+ \preceq e^-$ such that

$$\mathcal{S}(\mathbf{I}) = \{\sup_r(I) \mid I \in \mathcal{I}(P^*)\}.$$

We refer to a map r appearing in the property (R) as a *reduction map*. In other words, the property (R) requires that every instance $\mathbf{I}$ of Prob admits a reduction map r such that $\sup_r$ forms a surjection from the family $\mathcal{I}(P^*)$ of ideals of P^* to the family $\mathcal{S}(\mathbf{I})$ of solutions of $\mathbf{I}$. A toy example of a reduction map is found in Example 1.

▶ **Example 1.** Let $E = \{a, b, c, d\}$ be the ground set. Suppose that the set of solutions is $\mathcal{S}(\mathbf{I}) = \{\{a, b\}, \{b, c\}, \{c, d\}\}$. We consider a poset $(P = \{\bot, u, v, \top\}, \preceq)$, where $\bot$ is the minimum element, $\top$ is the maximum element, and $u \prec v$. We define a map $r\colon E \to P^2$ by $r(a) = (\bot, u)$, $r(b) = (\bot, v)$, $r(c) = (u, \top)$, $r(d) = (v, \top)$. Since $\mathcal{I}(P^*) = \{\emptyset, \{u\}, \{u, v\}\}$, we can see that r is a reduction map. Indeed, we have $\sup_r(\emptyset) = \{e \in E \mid e^+ \in \{\bot\} \not\ni e^-\} = \{a, b\}$, $\sup_r(\{u\}) = \{b, c\}$, and $\sup_r(\{u, v\}) = \{c, d\}$.

We show that if the problem Prob has the properties (S) and (R), then $\mathcal{S}(\mathbf{I})$ with a partial order defined from a reduction map forms a distributive lattice (see Theorem 11).

Our framework exploits network flow techniques to efficiently compute diverse solutions. To utilize them, we will construct a directed acyclic graph (DAG) to represent a poset. Here, we say that a DAG G *represents* a poset $(P, \preceq)$ with distinct minimum element $\bot$ and maximum element $\top$ if the vertex set of G is P, $\top$ is a source and $\bot$ is a sink in G, and for any $u, v \in P^*$, there is a u-v path in G if and only if $u \succeq v$.

Our main result is stated as follows. We focus on deterministic algorithms in this paper.

▶ **Theorem 2.** *Suppose that a combinatorial problem* Prob *has the properties (S) and (R). Then, for any instance* $\mathbf{I}$ *of* Prob*, the problems* Sum-k-Diverse *and* Cov-k-Diverse *of* Prob *can be solved in* $O(T_P(\mathbf{I}) + T_r(\mathbf{I}) + (|A| + k|E|)^{1+o(1)})$ *time and in* $O(T_P(\mathbf{I}) + T_r(\mathbf{I}) + (|A| + |E|)^{1+o(1)} \log^2 k + kq)$ *time, respectively. Here,* E *is the ground set of* $\mathbf{I}$*,* q *is the size of each solution of* $\mathbf{I}$*,* $T_P(\mathbf{I})$ *is the time required to construct a DAG representing the poset* P *in (R),* $T_r(\mathbf{I})$ *is the time required to construct a reduction map* r *in (R), and* A *is the set of arcs of a DAG* G *that represents* P*.*

We obtain Theorem 2 via the reduction of Sum-k-Diverse/Cov-k-Diverse of Prob to a classical network flow problem, called the *minimum cost flow problem*. This enables us to utilize the state-of-the-art (deterministic) algorithm [25] for the minimum cost flow problem in solving Sum-k-Diverse/Cov-k-Diverse of Prob. To this end, we introduce an intermediate problem called the *minimum k-potential problem*, and reduce the k-diverse problems to the minimum k-potential problem. Then we further reduce the minimum k-potential problem to the minimum cost flow problem by utilizing the idea of Ahuja, Hochbaum, and Orlin [1], who dealt with a more general problem. Our novelty is to identify the properties (S) and (R) as a sufficient condition to reduce the k-diverse problem to the minimum k-potential problem.

We also develop a reduction of Sum-k-Diverse/Cov-k-Diverse of Prob to the *maximum s-t flow problem* via the minimum k-potential problem. We describe this in the full version. While the running-time of this reduction is slightly worse than that of the reduction to the minimum cost flow problem, its practical performance may be superior thanks to the simplicity of the reduction and the maximum s-t flow problem.

As applications of our framework, we demonstrate that two classical combinatorial problems, Unweighted Minimum s-t Cut and Stable Matching, have properties (S) and (R). Here, Unweighted Minimum s-t Cut is to find an s-t cut of a given digraph G with the minimum size, and Stable Matching is to find a matching of two parties such that no unmatched pair both prefer each other to their current partners (see Sections 5.2 and 5.3 for the definitions). In fact, both problems are known to possess a distributive lattice structure. By applying Theorem 2 to Sum-k-Diverse/Cov-k-Diverse of these problems, we obtain the following results.

▶ **Theorem 3.**

1. *The problems* Sum-k-Diverse *and* Cov-k-Diverse *of* Unweighted Minimum s-t Cut *can be solved in* $O(n + (km)^{1+o(1)})$ *time and in* $O(n + m^{1+o(1)} \log^2 k + kq)$ *time, respectively, where* n *denotes the number of vertices of the input digraph* G*,* m *the number of arcs of* G*, and* q *the size of a minimum* s-t *cut of* G*.*
2. *The problems* Sum-k-Diverse *and* Cov-k-Diverse *of* Stable Matching *can be solved in* $O((kn^2)^{1+o(1)})$ *time and in* $O(n^{2+o(1)} \log^2 k + kn)$ *time, respectively, where* n *denotes the size of the ground set* U *(or* V*) of the input instance* $(U, V; (\leq_u)_{u \in U}, (\leq_v)_{v \in V})$*.*

The polynomial-time solvability of Sum-k-Diverse/Cov-k-Diverse of Unweighted Minimum s-t Cut has already been shown by De Berg, Martínez, and Spieksma [7]. Very recently, independently of our work, the same authors [8] developed a framework for solving

Sum-k-Diverse/Cov-k-Diverse of a combinatorial problem having a certain property, and they show that Sum-k-Diverse/Cov-k-Diverse of Stable Matching can be solved in polynomial time. Both of their tractability results in [7, 8] are based on the polynomial-time solvability of the *submodular function minimization (SFM)* [2, 12]. Even if we use the state-of-the-art algorithm for SFM, given in [18], the running-time of their algorithms is (polynomial but) not very fast; it takes $O(k^5n^5)$ time for Sum-k-Diverse/Cov-k-Diverse of Unweighted Minimum s-t Cut, and $O(k^5n^9)$ time for that of Stable Matching. As in Theorem 3, our proposed algorithms for Sum-k-Diverse/Cov-k-Diverse of Unweighted Minimum s-t Cut/Stable Matching are much faster, thanks to recent advance in network flow algorithms. Moreover, we show that the framework of [8] for Sum-k-Diverse and Cov-k-Diverse can be captured by our framework (see Section 5.4). This allows us to improve the time complexity.

Due to the space limitation, all the proofs (marked with $\star$) are omitted, and they can be found in the full version.

Related work. There exists a vast body of literature on the problem of finding diverse solutions; here, we mention only a few representative papers. Finding diverse solutions is generally harder than finding a single one. The k-diverse problems with respect to d_{sum} of some polynomial-time solvable problems – such as the maximum matching problem [10] and the (global) minimum cut problem [15] – are known to be NP-hard. Thus, in recent years, there has been a growing body of work on fixed-parameter tractable (FPT) algorithms for NP-hard diverse problems [3, 4, 9, 10, 11, 17]. Meanwhile, polynomial-time algorithms are known for several other k-diverse problems with respect to the diversity measures d_{sum} or d_{cov}, including the k-diverse variants of the spanning-tree problem [17], the shortest s-t path problem [16], as well as Unweighted Minimum s-t Cut [7] and Stable Matching [8], which have already been mentioned.

2 Preliminaries

Let $\mathbb{Z}$, $\mathbb{Z}_+$, $\mathbb{R}$, and $\mathbb{R}_+$ denote the set of integers, nonnegative integers, real numbers, and nonnegative real numbers, respectively. For a nonnegative integer k, let $[k] \coloneqq \{1, 2, \ldots, k\}$ and $[0, k] \coloneqq \{0, 1, \ldots, k\}$. The symmetric difference $(S \setminus T) \cup (T \setminus S)$ of sets S and T is denoted by $S \triangle T$. For a finite set E and a nonnegative integer $q \in \mathbb{Z}_+$, let $\binom{E}{q}$ denote the family of all subsets of E of size q, i.e., $\binom{E}{q} \coloneqq \{S \subseteq E \mid |S| = q\}$. For any finite set E, element $e \in E$, and k-tuple $\mathbf{S} = (S_1, S_2, \ldots, S_k)$ of subsets of E, the *multiplicity* of e with respect to $\mathbf{S}$, denoted by $\mu_e(\mathbf{S})$, is defined as the number of subsets S_i that contain e, i.e., $\mu_e(\mathbf{S}) \coloneqq |\{i \in [k] \mid e \in S_i\}|$.

Posets. A *partially ordered set* (or *poset*) is a pair $(P, \preceq)$ of a set P and a binary relation $\preceq$ over P satisfying, for $x, y, z \in P$, that $x \preceq x$ (reflexivity), $x \preceq y$ and $y \preceq x$ imply $x = y$ (antisymmetry), and $x \preceq y$ and $y \preceq z$ imply $x \preceq z$ (transitivity). By $x \prec y$ we mean $x \preceq y$ and $x \neq y$. Such a binary relation is called a *partial order*. If no confusion arises, we denote by P a poset and its underlying set interchangeably. In this paper, we consider only a finite poset, i.e., posets whose underlying sets are finite. Hence, by a poset we mean a finite poset. A partial order $\preceq$ on P is called a *total order* if $x \preceq y$ or $y \preceq x$ holds for any $x, y \in P$. For a poset $(P, \preceq)$, a subset $I \subseteq P$ is called an *ideal* if I is closed under $\preceq$, i.e., for any $v \in P$ and $u \preceq v$, we have $u \in P$. Let $\mathcal{I}(P)$ denote the set of all ideals of P. If P has the minimum element $\bot$ and the maximum element $\top$, then we denote by P^* the poset obtained from P by removing $\bot$ and $\top$.

Network flows. Let $G = (V, A)$ be a digraph. For a vertex subset $X \subseteq V$, let $\Delta^+_G(X)$ (resp. $\Delta^-_G(X)$) denote the set of outgoing arcs from (resp. incoming arcs to) X. If no confusion arises, we omit the subscript G from $\Delta^+_G(X)$ and $\Delta^-_G(X)$. If X consists of a single vertex v, i.e., $X = \{v\}$, we simply write $\Delta^+(v)$ and $\Delta^-(v)$ instead of $\Delta^+(\{v\})$ and $\Delta^-(\{v\})$, respectively. A function $f\colon A \to \mathbb{Z}_+$ from the arc set A to the nonnegative integers is called a *flow* of G. In the case where the digraph G has an arc capacity $c\colon A \to \mathbb{Z}_+$, a flow $f\colon A \to \mathbb{Z}_+$ is said to be *feasible* (with respect to c) if $f(a) \leq c(a)$ for all $a \in A$. For a feasible flow f with respect to an arc capacity c, its *residual graph*, denoted by G_f, is the digraph whose vertex set is V and arc set is $\{a \mid a \in A,\ f(a) < c(a)\} \cup \{a \mid \bar{a} \in A,\ 0 < f(\bar{a})\}$, where $\bar{a}$ denotes the reverse arc (v, u) of $a = (u, v)$. This plays an important role in algorithms for network flow problems (and our algorithms). For a flow $f\colon A \to \mathbb{Z}_+$, its *boundary* $\partial f\colon V \to \mathbb{Z}$ is defined by $\partial f(v) \coloneqq \sum_{a \in \Delta^+(v)} f(a) - \sum_{a \in \Delta^-(v)} f(a)$ for each $v \in V$.

In the following (except for Section 5.4), we assume that Prob is a combinatorial problem having the properties (S) and (R). For an instance $\mathbf{I}$ of Prob, we denote by $\mathcal{S}(\mathbf{I})$ the family of solutions of $\mathbf{I}$, $T_P(\mathbf{I})$ the time required to construct a DAG representing the poset P in (R), and $T_r(\mathbf{I})$ the time required to construct a reduction map r in (R).

3 Reduction to the minimum k-potential problem

In this section, we provide a reduction from the k-diverse problem of Prob to the *minimum k-potential problem*, which we introduce later.

In Sum-k-Diverse and Cov-k-Diverse of Prob, we can regard the diversity measures d_{sum} and d_{cov} as the functions over $(\mathcal{S}(\mathbf{I}))^k$ for each instance $\mathbf{I}$ of Prob. Since $\mathcal{S}(\mathbf{I}) \subseteq \binom{E}{q}$ holds by the property (S), we have $\sum_{e \in E} \mu_e(\mathbf{S}) = \sum_{i=1}^k |S_i| = kq$ for any $\mathbf{S} = (S_1, S_2, \ldots, S_k) \in (\mathcal{S}(\mathbf{I}))^k$, which is a constant. Hence, the functions d_{sum} and d_{cov} are representable as

$$d_{\mathrm{sum}}(\mathbf{S}) = \sum_{e \in E} \mu_e(\mathbf{S})(k - \mu_e(\mathbf{S})) = \text{Const.} - \sum_{e \in E} \mu_e(\mathbf{S})^2,$$
$$d_{\mathrm{cov}}(\mathbf{S}) = \sum_{e \in E} \min\{1, \mu_e(\mathbf{S})\} = \text{Const.} - \sum_{e \in E} \max\{0, \mu_e(\mathbf{S}) - 1\}$$

for each $\mathbf{S} \in (\mathcal{S}(\mathbf{I}))^k$. Thus, the problems Sum-k-Diverse and Cov-k-Diverse of Prob, namely, the problems of maximizing the functions d_{sum} and d_{cov} over $(\mathcal{S}(\mathbf{I}))^k$, are equivalent to those of minimizing

$$d^*_{\mathrm{sum}}(\mathbf{S}) \coloneqq \sum_{e \in E} \mu_e(\mathbf{S})^2, \qquad d^*_{\mathrm{cov}}(\mathbf{S}) \coloneqq \sum_{e \in E} \max\{0, \mu_e(\mathbf{S}) - 1\},$$

respectively.

By using the concept of discrete convex functions, we can uniformly handle these functions d^*_{sum} and d^*_{cov} as follows. A function $\varphi\colon \mathbb{Z} \to \mathbb{Z}$ is said to be *discrete convex* [20, Chapter 3.4] if $\varphi(x-1) + \varphi(x+1) \geq 2\varphi(x)$ for all $x \in \mathbb{Z}$, and said to be *non-decreasing on* $\mathbb{Z}_+$ if $\varphi(x) \leq \varphi(x+1)$ for all $x \in \mathbb{Z}_+$. For a discrete convex function φ with $\varphi(0) = 0$ that is non-decreasing on $\mathbb{Z}_+$, we define

$$d^*_\varphi(\mathbf{S}) \coloneqq \sum_{e \in E} \varphi(\mu_e(\mathbf{S}))$$

for $\mathbf{S} \in (\mathcal{S}(\mathbf{I}))^k$. Since the functions $x \mapsto x^2$ and $x \mapsto \max\{0, x-1\}$ are discrete convex functions that are non-decreasing on $\mathbb{Z}_+$ and satisfy $0 \mapsto 0$, both of d^*_{sum} and d^*_{cov} admit such representations.

Our framework can be applied to the k-diverse problem with respect to the diversity measure of the form

$$d_\varphi(\mathbf{S}) \coloneqq \text{Const.} - d^*_\varphi(\mathbf{S})$$

for $\mathbf{S} \in (\mathcal{S}(\mathbf{I}))^k$, where φ is a discrete convex function with $\varphi(0) = 0$ that is non-decreasing on $\mathbb{Z}_+$. In the following, we consider the k-diverse problem with respect to d_φ of Prob, or equivalently, the problem of minimizing d^*_φ over $(\mathcal{S}(\mathbf{I}))^k$ for a discrete convex function φ.

We then introduce the *minimum k-potential problem*, to which we reduce the problem of minimizing d^*_φ later. Let $G = (V, A)$ be a DAG having unique source vertex $\top$ and sink vertex $\bot$ with $\top \neq \bot$. We refer to an assignment $p\colon V \to \mathbb{Z}_+$ of integers to vertices as a *k-potential* if p satisfies the following conditions:

(P1) $p(\bot) = k$ and $p(\top) = 0$.

(P2) $0 \leq p(v) \leq k$ for each $v \in V$.

(P3) p is monotone non-increasing with respect to A, i.e., $p(u) \leq p(v)$ for each $(u, v) \in A$.

In the *minimum k-potential problem*, we are given a DAG $G = (V, A)$ with unique source $\top$ and sink $\bot$, an arc weight $w\colon A \to \mathbb{Z}_+$, and a discrete convex function $\varphi\colon \mathbb{Z} \to \mathbb{Z}$ with $\varphi(0) = 0$ that is non-decreasing on $\mathbb{Z}_+$, and asked to find a k-potential p of G that minimizes

$$H(p) \coloneqq \sum_{a=(u,v)\in A} w(a)\varphi(p(v) - p(u)).$$

We note that the function H does not change even if we remove vertices $v \in V \setminus \{\bot, \top\}$ with $\Delta^+(v) = \Delta^-(v) = \emptyset$, called *isolated vertices*, from G.

Finally, we reduce, for an instance $\mathbf{I}$ of Prob, the problem of minimizing d^*_φ over $(\mathcal{S}(\mathbf{I}))^k$ to the minimum k-potential problem by utilizing the property (R). Let $(P, \preceq)$ be a poset having minimum element $\bot$ and maximum element $\top$, and let $r\colon E \to P^2$ be a reduction map as in the property (R). Then, we construct a DAG $G_\mathbf{I}$ whose vertex set is $V = P$ and whose arc set is $A = A_P \cup A_E$, where A_P is an arc set such that a DAG (P, A_P) represents the poset $(P, \preceq)$, and $A_E \coloneqq \{(e^-, e^+) \mid e \in E\}$. The resulting $G_\mathbf{I}$ is still a DAG that represents P and has unique source $\top$ and sink $\bot$; each arc $(e^-, e^+) \in A_E$ is compatible with the partial order $\preceq$ of P, since $e^- \succeq e^+$. An arc weight $w_\mathbf{I}\colon A \to \mathbb{Z}_+$ is defined by $w_\mathbf{I}(a) \coloneqq |\{e \in E \mid a = (e^-, e^+)\}|$ for $a \in A$. Note that $w_\mathbf{I}(a) = 0$ for each $a \in A \setminus A_E$. We set φ, which satisfies the non-decreasing property on $\mathbb{Z}_+$ and $\varphi(0) = 0$, as the input discrete convex function of the minimum k-potential problem. Then, the triple $(G_\mathbf{I}, w_\mathbf{I}, \varphi)$ is an instance of the minimum k-potential problem; its construction time is $|E|$.

▶ **Example 4.** Recall the toy example in Example 1. A graph $G = (P = \{\bot, u, v, \top\}, A_P)$ with $A_P = \{(u, \bot), (v, u), (\top, v)\}$ is a DAG representing $(P, \preceq)$. By construction, $A_E = \{(u, \bot), (v, \bot), (\top, u), (\top, v)\}$. Then, the DAG $G_\mathbf{I} = (P, A_P \cup A_E)$ is illustrated in Figure 1. The arc weights are $w_\mathbf{I}(v, u) = 0$, and $w_\mathbf{I}(a) = 1$ for all other arcs $a \in (A_P \cup A_E) \setminus \{(v, u)\}$.

Intuitively, any k-potential represents the direct sum of k ideals $I_1, I_2, \ldots, I_k \in \mathcal{I}(P^*)$ as a multiset. Conversely, for any k ideals in $\mathcal{I}(P^*)$, there exists a k-potential that represents their direct sum as a multiset. Furthermore, by the property (R), each ideal of P^* corresponds to a solution of $\mathbf{I}$ via $\sup_r$. The following lemma verifies this intuition.

▶ **Lemma 5 ($\star$).** *For each k-tuple $\mathbf{S} \in (\mathcal{S}(\mathbf{I}))^k$, there is a k-potential $p_\mathbf{S}$ of $G_\mathbf{I}$ such that $H(p_\mathbf{S}) = d^*_\varphi(\mathbf{S})$. Conversely, for each k-potential p of $G_\mathbf{I}$, there is a k-tuple $\mathbf{S}_p \in (\mathcal{S}(\mathbf{I}))^k$ of solutions of $\mathbf{I}$ such that $H(p) = d^*_\varphi(\mathbf{S}_p)$, and we can construct $\mathbf{S}_p$ from p in $O(|E| + kq)$ time.*

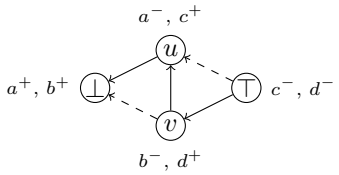


Figure 1 A DAG $G_{\mathbf{I}}$ for the toy example. Arcs in A_P and $A_E \setminus A_P$ are shown as solid and dashed lines, respectively.

Lemma 5 immediately implies that we can construct in $O(|E| + kq)$ time a minimizer $\mathbf{S}_p$ of d^*_φ over $(\mathcal{S}(\mathbf{I}))^k$ from a minimum k-potential p for the instance $(G_{\mathbf{I}}, w_{\mathbf{I}}, \varphi)$ of the minimum k-potential problem. Therefore, we obtain the following.

▶ **Theorem 6** (⋆). *We can solve the k-diverse problem with respect to d_φ of* Prob *in $O(T_P(\mathbf{I}) + T_r(\mathbf{I}) + T_{\mathrm{mp}}(\mathbf{I}) + |E| + kq)$ time, where $\mathbf{I}$ is a given instance of* Prob *and $T_{\mathrm{mp}}(\mathbf{I})$ is the time of solving the instance of the minimum k-potential problem reduced from $\mathbf{I}$.*

4 Solving the minimum k-potential problem via minimum cost flow

Here, we reduce the minimum k-potential problem to the minimum cost flow problem.

Let us first recall the *minimum cost flow problem* (see e.g., [19, Chapter 9] and [23, Chapter 12] for details). In this problem, we are given a digraph $G = (V, A)$, arc cost $\gamma\colon A \to \mathbb{Z}$, arc capacity $c\colon A \to \mathbb{Z}_+$, and vertex demand $d\colon V \to \mathbb{Z}$, and asked to find a feasible flow $f\colon A \to \mathbb{Z}_+$ (i.e., $f(a) \le c(a)$ for $a \in A$) that minimizes $\sum_{a\in A} \gamma(a)f(a)$ subject to $\partial f(v) = d(v)$ for all $v \in V$. An optimal solution of the minimum cost flow problem is called a *minimum cost flow*. We denote by $T_{\mathrm{mcf}}(n, m, \Gamma, C, D)$ the time required to solve the minimum cost flow problem for a network of n vertices and m arcs with cost at most Γ in absolute values, capacity at most C, and a demand vector with values at most D in absolute values. Using the state-of-the-art algorithm for the minimum cost flow problem given in [25], we have $T_{\mathrm{mcf}}(n, m, \Gamma, C, D) = O(m^{1+o(1)} \log(\max\{C, D\}) \log \Gamma)$.

Our reduction follows the work by Ahuja, Hochbaum, and Orlin [1], who dealt with a more general problem called the *convex cost integer dual network flow problem*. They showed that the Lagrangian dual of their problem is reduced to the minimum cost flow problem. For the complexity analysis, we explicitly provide the reduction in our case and the construction of a minimum k-potential from a minimum cost flow.

For a discrete convex function $\psi\colon \mathbb{Z} \to \mathbb{Z}$, an integer $x \in \mathbb{Z}$ is called a *breakpoint* of ψ if $\psi(x+1)+\psi(x-1) > 2\psi(x)$, i.e., the *left slope* $\psi(x)-\psi(x-1)$ and *right slope* $\psi(x+1)-\psi(x)$ of ψ at x are different. We can observe that, for each $x \in \mathbb{Z}$, the left slope of ψ at x is at most the right slope of ψ at x. Let $B(\psi)$ denote the set of breakpoints of ψ.

Let $(G = (V, A), w\colon A \to \mathbb{Z}_+, \varphi\colon \mathbb{Z} \to \mathbb{Z})$ be an instance of the minimum k-potential problem. We define $B_k(\varphi) := (B(\varphi) \cap [0, k]) \cup \{0, k\}$ and suppose that $B_k(\varphi) = \{b_0, b_1, \dots, b_z\}$ with $(0 =)b_0 < b_1 < \cdots < b_z (= k)$. Let s_i^- (resp. s_i^+) denote the left (resp. right) slope of φ at $b_i \in B(\varphi) \cap [k-1]$; note that $s_i^+ = s_{i+1}^- < s_{i+1}^+$. We set $M \in \mathbb{Z}_+$ as a sufficiently large integer satisfying $M > H(p)$ for any k-potential p of G, e.g., $M = \sum_{a\in A} w(a)\varphi(k) + 1$.

We construct an instance of the minimum cost flow problem. The vertex set of the input digraph $\bar{G}$ is $\bar{V} := V \cup \{0\}$. We set the arc set $\bar{A}$ of $\bar{G}$, arc cost $\bar{\gamma}\colon \bar{A} \to \mathbb{Z}$, and capacity $\bar{c}\colon \bar{A} \to \mathbb{Z}_+$ by creating

- $|B_k(\varphi)| = z+1$ copies of each $a \in A$ satisfying $w(a) > 0$ with costs $b_0, b_1, \dots, b_{z-1}, b_z$ and capacities $w(a)s_1^- + M, w(a)(s_1^+ - s_1^-), w(a)(s_2^+ - s_2^-), \dots, w(a)(s_{z-1}^+ - s_{z-1}^-), M - w(a)s_{z-1}^+$, respectively,

- two copies of each $a \in A$ satisfying $w(a) = 0$ or each $a = (0, u)$ for $u \in V \setminus \{\bot, \top\}$ with costs $0, k$ and capacities M, M, respectively,
- arcs $(0, \bot)$ and $(0, \top)$ with cost k and 0, respectively, and capacity $2M$.

The vertex demand $\bar{d}\colon \bar{V} \to \mathbb{Z}$ is set as $\bar{d}(v) \coloneqq M(|\Delta^+_{G'}(v)| - |\Delta^-_{G'}(v)|)$ for each $v \in \bar{V}$, where $G' \coloneqq (\bar{V}, A')$ and $A' \coloneqq A \cup \{(0, v) \mid v \in V\}$.

Let f^* be a minimum cost flow of the resulting instance $(\bar{G}, \bar{\gamma}, \bar{c}, \bar{d})$. Then we construct the residual graph $\bar{G}_{f^*}$ of $\bar{G}$ with respect to f^*, and set the arc length ℓ of $\bar{G}_{f^*}$ as $\ell(a) \coloneqq \gamma(a)$ if $a \in \bar{A}$, and $\ell(a) \coloneqq -\gamma(a)$ if $\bar{a} \in \bar{A}$. Let us define $\bar{p}^*\colon \bar{V} \to \mathbb{Z}$ as a feasible potential with $\bar{p}^*(0) = 0$ in $\bar{G}_{f^*}$ with respect to arc length ℓ, i.e., an assignment $\bar{p}^*\colon \bar{V} \to \mathbb{Z}$ satisfying $\bar{p}^*(0) = 0$ and $\ell(a) \geq \bar{p}^*(v) - \bar{p}^*(u)$ for each arc $a = (u, v) \in \bar{A}$. The following lemma justifies our reduction.

▶ **Lemma 7** (⋆). *The restriction $p^*\colon V \to \mathbb{Z}$ of $\bar{p}^*$ to V forms a minimum k-potential of (G, w, φ).*

The validity of Lemma 7 follows from exactly the same argument given in [1]. For the sake of completeness, we provide a direct proof in the full version.

We then show the time complexity of solving the minimum k-potential problem via our reduction. The digraph $\bar{G}$ has $|V| + 1 = O(|V|)$ vertices, and at most $2(|V| - 1 + |A_0|) + |B_k(\varphi)||A_+| = O(|V| + |A_0| + |B_k(\varphi)||A_+|)$ arcs. Here, A_0 and A_+ denote the set of arcs a with $w(a) = 0$ and $w(a) > 0$, respectively. Thus, we can construct the minimum cost flow instance $(\bar{G}, \bar{\gamma}, \bar{c}, \bar{d})$ in $O(|V| + |A_0| + |B_k(\varphi)||A_+|)$ time. The costs and capacities are nonnegative integers at most k and $\bar{C} \coloneqq \max\{2M, \max_{a \in A} w(a)\varphi(k) + M\}$, respectively. The demands are at most $M|V|$ in absolute values. Hence, we can find a minimum cost flow $f^*\colon \bar{A} \to \mathbb{Z}_+$ of $(\bar{G}, \bar{\gamma}, \bar{c}, \bar{d})$ in $O\big(T_{\mathrm{mcf}}(|V|, |V| + |A_0| + |B_k(\varphi)||A_+|, k, \bar{C}, M|V|)\big)$ time. The residual graph $\bar{G}_{f^*}$ of $\bar{G}$ with respect to f^* is constructed in $O(|\bar{A}|) = O(|V| + |A_0| + |B_k(\varphi)||A_+|)$ time. A feasible potential $\bar{p}^*$ in $\bar{G}_{f^*}$ with respect to ℓ is found in $O(T_{\mathrm{mcf}}(|V|, |V| + |A_0| + |B_k(\varphi)||A_+|, k, |V|, |V|))$ time by computing shortest distances from a supernode to vertices in $\bar{V}$. Therefore, by Lemma 7, we can obtain a minimum k-potential p^*, which is a restriction of $\bar{p}^*$ to V, in $O\big(T_{\mathrm{mcf}}(|V|, |V| + |A_0| + |B_k(\varphi)||A_+|, k, \bar{C}, M|V|)\big)$ time.

By applying the algorithm in [25], we obtain the following, which implies Theorem 2.

▶ **Theorem 8** (⋆). *We can solve the k-diverse problem with respect to d_φ of* Prob *for an instance $\mathbf{I}$ in $O(T_P(\mathbf{I}) + T_r(\mathbf{I}) + (|A_P| + |B_k(\varphi)||E|)^{1+o(1)} \log(\varphi(k)) \log k + kq)$ time. In particular, the problems* Sum-k-Diverse *of* Prob *and* Cov-k-Diverse *of* Prob *can be solved in $O(T_P(\mathbf{I}) + T_r(\mathbf{I}) + (|A_P| + k|E|)^{1+o(1)})$ time and in $O(T_P(\mathbf{I}) + T_r(\mathbf{I}) + (|A_P| + |E|)^{1+o(1)} \log^2 k + kq)$ time, respectively.*

5 Applications

In this section, we introduce two applications of our framework; one is the k-diverse problems of Unweighted Minimum s-t Cut and the other is that of Stable Matching.

In order to apply our framework to the k-diverse problem of a concrete combinatorial problem, we need to construct a poset and a reduction map appearing in the property (R) for each instance. In Section 5.1, we develop a common strategy of building these components. In fact, it is known that both families of minimum s-t cuts and stable matchings are naturally identified with set families called *ring families*. A ring family with inclusion order forms a poset, particularly a *distributive lattice*. We will utilize these facts to construct a reduction map for Unweighted Minimum s-t Cut and Stable Matching in Sections 5.2 and 5.3,

respectively. In Section 5.4, we briefly describe a framework for the k-diverse problem recently developed by De Berg, Martínez, and Spieksma in [8], and show that our framework can capture theirs in the case of Sum-k-Diverse/Cov-k-Diverse.

5.1 How to construct a poset and a reduction map

We start this subsection with introducing terminology on lattices (see e.g., [6] for details). A poset $(L, \preceq)$ is called a *lattice* if, for any two elements $x, y \in L$, their least upper bound (*join*) $x \vee y$ and greatest lower bound (*meet*) $x \wedge y$ exist in L. A lattice L is said to be *distributive* if the distributive law $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$ holds for any $x, y, z \in L$.

A typical example of a distributive lattice is the family $\mathcal{I}(P)$ of ideals of a poset over P with inclusion order $\subseteq$. In this case, the join of two ideals I and I' is their union $I \cup I'$, and the meet is their intersection $I \cap I'$; we can easily see that they are also ideals. More generally, a *ring family*, which is a nonempty family $\mathcal{R} \subseteq 2^R$ of subsets of a nonempty finite set R such that it is closed under the union and intersection, endowed with inclusion order $\subseteq$ forms a distributive lattice $(\mathcal{R}, \subseteq)$, where $X \vee Y = X \cup Y$ and $X \wedge Y = X \cap Y$ for $X, Y \in \mathcal{R}$.

The celebrated *Birkhoff's representation theorem* [5] states that every distributive lattice $(L, \preceq)$ is isomorphic to the distributive lattice $(\mathcal{I}(P), \subseteq)$ for some poset P. Here, two lattices $(L, \preceq)$ and $(L', \preceq')$ are said to be *isomorphic* if there is a bijection $h\colon L \to L'$ such that $x \preceq y$ if and only if $h(x) \preceq h(y)$ for any $x, y \in L$. For a distributive lattice $(L, \preceq)$, we refer to a poset P such that $(L, \preceq)$ and $(\mathcal{I}(P), \subseteq)$ are isomorphic as a *Birkhoff representation* of $(L, \preceq)$.

A Birkhoff representation of the distributive lattice $(\mathcal{R}, \subseteq)$ over a ring family $\mathcal{R}$ can be obtained as follows. The ring family $\mathcal{R}$ has the unique minimal set $X_\bot := \bigcap_{X \in \mathcal{R}} X$ and unique maximal set $X_\top := \bigcup_{X \in \mathcal{R}} X$. Take any maximal chain $X_\bot =: X_0 \subsetneq X_1 \subsetneq \cdots \subsetneq X_n := X_\top$ from $X_\bot$ to $X_\top$ in $\mathcal{R}$; namely, there are no $X \in \mathcal{R}$ and $i \in [n]$ with $X_{i-1} \subsetneq X \subsetneq X_i$. Then, $\Pi^*(\mathcal{R}) := \{X_i \setminus X_{i-1} \mid i \in [n]\}$ forms a partition of $X_\top \setminus X_\bot$. We define a partial order $\preceq$ on $\Pi^*(\mathcal{R})$ by setting $\hat{X} \preceq \hat{Y}$ if and only if every $Z \in \mathcal{R}$ with $Z \supseteq \hat{Y}$ also includes $\hat{X}$. The resulting $(\Pi^*(\mathcal{R}), \preceq)$ is actually a poset and is independent of the choice of a maximal chain from $X_\bot$ to $X_\top$. It is known (see e.g., [21, Chapter 2.2.2]) that $(\Pi^*(\mathcal{R}), \preceq)$ is a Birkhoff representation of $(\mathcal{R}, \subseteq)$; more precisely, the map $\mathcal{Y} \mapsto (\bigcup_{\hat{X} \in \mathcal{Y}} \hat{X}) \cup X_\bot$ is an isomorphism from $(\mathcal{I}(\Pi^*(\mathcal{R})), \subseteq)$ to $(\mathcal{R}, \subseteq)$. In this paper, we refer to $(\Pi^*(\mathcal{R}), \preceq)$ as *the* Birkhoff representation of $(\mathcal{R}, \subseteq)$.

We are ready to develop a strategy to construct a reduction map by using a ring family. Let $\mathcal{R} \subseteq 2^R$ be a ring family over a nonempty finite set R. We may assume that the minimal set $X_\bot$ is nonempty and the maximal set $X_\top$ is a proper subset of R; otherwise we add two elements $\bot$ and $\top$ to R and update each subset $X \in \mathcal{R}$ as $X \cup \{\bot\}$, which makes $\mathcal{R}$ satisfy $\bot \in X_\bot$ and $\top \in \overline{X}_\top := R \setminus X_\top$. Let $\Pi(\mathcal{R})$ denote the partition $\Pi^*(\mathcal{R}) \cup \{X_\bot, \overline{X}_\top\}$ of R. We extend the partial order $\preceq$ on $\Pi^*(\mathcal{R})$ to that on $\Pi(\mathcal{R})$ by setting $X_\bot \prec \hat{X} \prec \overline{X}_\top$ for any $\hat{X} \in \Pi^*(\mathcal{R})$. For a map $\hat{r}\colon E \to R^2$ given by $e \mapsto (\hat{e}^+, \hat{e}^-)$, we define

$$\sup_{\hat{r}}(X) := \{e \in E \mid \hat{e}^+ \in X \not\ni \hat{e}^-\}$$

for $X \in \mathcal{R}$. We say that $\hat{r}\colon E \to R^2$ is a *pre-reduction map* if $\hat{e}^- \in X$ implies $\hat{e}^+ \in X$ for all $X \in \mathcal{R}$ and $e \in E$, and $\mathcal{S}(\mathbf{I}) = \{\sup_{\hat{r}}(X) \mid X \in \mathcal{R}\}$. We show that we can construct a reduction map as long as we have a pre-reduction map.

▶ **Lemma 9** (⋆). *Suppose that $\hat{r}\colon E \to R^2$ given by $e \mapsto (\hat{e}^+, \hat{e}^-)$ is a pre-reduction map. Then, the map $r\colon E \to \Pi(\mathcal{R})^2$ defined by*

$$r(e) := (\Pi(\hat{e}^+), \Pi(\hat{e}^-)) \qquad (e \in E)$$

is a reduction map, where $\Pi(x)$ $(x \in R)$ denotes the unique member of $\Pi(\mathcal{R})$ that contains x.

Thus, we obtain the following:

▶ **Proposition 10.** *Let R be a nonempty finite set, and let $\mathcal{R}$ be a ring family over R. Suppose we are given a pre-reduction map $\hat{r}\colon E \to R^2$ and the partition $\Pi(\mathcal{R})$ of R. Then, we can construct a reduction map $r\colon E \to \Pi(\mathcal{R})^2$ in constant time.*

The property (R) requires that, for each instance $\mathbf{I}$ of Prob, the family $\mathcal{S}(\mathbf{I})$ of its solutions is closely related to the family $\mathcal{I}(P^*)$ of ideals of the poset P^*. Hence, it is natural to expect that $\mathcal{S}(\mathbf{I})$ admits the distributive lattice structure. We conclude this subsection by showing that this indeed holds, which might be viewed as a necessary condition on Prob for our framework to be applicable.

Recall the notation in the property (R). We may assume that $e^+ \prec e^-$ for each $e \in E$, since if $e^+ = e^-$, then no member S in $\mathcal{S}(\mathbf{I})$ contains e, i.e., $\mathcal{S}(\mathbf{I}) \subseteq \binom{E\setminus\{e\}}{q}$, and hence we can remove e from E. Let $\leq$ denote a partial order on E such that $e \leq e'$ if and only if $e = e'$ or $e^- \preceq e'^+$. This $\leq$ can be extended for $\mathcal{S}(\mathbf{I})$ by setting $S \leq T$ if and only if there exists a bijection $\pi\colon S \to T$ such that $e \leq \pi(e)$ for all $e \in S$. Then, the following holds.

▶ **Theorem 11 ($\star$).** *Suppose that a combinatorial problem Prob has the properties (S) and (R). For any instance $\mathbf{I}$ of Prob, the poset $(\mathcal{S}(\mathbf{I}), \leq)$ forms a distributive lattice.*

5.2 The k-diverse unweighted minimum s-t cut problem

In this subsection, we consider the k-diverse problem of Unweighted Minimum s-t Cut. Here, this is the minimum s-t cut problem for a digraph $G = (V, A)$ with unit arc capacity $c\colon A \to \mathbb{Z}_+$, i.e., $c(a) = 1$ for all $a \in A$. Clearly, Unweighted Minimum s-t Cut has the property (S) by setting the ground set E as the arc set A and the integer q as the size of minimum s-t cuts. Our aim is to construct, for some ring family $\mathcal{R}$, a DAG D representing the Birkhoff representation $(\Pi^*(\mathcal{R}), \preceq)$, the partition $\Pi(\mathcal{R})$, and a pre-reduction map $\hat{r}$ based on $\mathcal{R}$, which implies that Unweighted Minimum s-t Cut has the property (R) by Lemma 9.

Let $G = (V, A)$ with $s, t \in V$ be a digraph that is an instance of Unweighted Minimum s-t Cut. Then, the family $\mathcal{S}(G) \subseteq 2^A$ of solutions of the instance G is the set of all minimum s-t cuts, which is a subset of the family $\{\Delta^+(X) \mid X \subseteq V \text{ with } s \in X \not\ni t\}$. Let $\mathcal{R}_{st}$ denote the family of vertex subsets $X \subseteq V$ with $s \in X \not\ni t$ such that its outgoing arcs form a minimum s-t cut of G, i.e.,

$$\mathcal{R}_{st} \coloneqq \{X \subseteq V \mid s \in X \not\ni t,\ \Delta^+(X) \in \mathcal{S}(G)\}.$$

It is well-known that $\mathcal{R}_{st}$ forms a ring family, which directly follows from Lemma 12 below. Moreover, since the unique minimal set $X_\perp$ in $\mathcal{R}_{st}$ contains s and the unique maximal set $X_\top$ in $\mathcal{R}_{st}$ excludes t, we have $X_\perp \neq \emptyset \neq \overline{X}_\top \coloneqq V \setminus X_\top$. We will construct a pre-reduction map based on $\mathcal{R}_{st}$.

We utilize the representation introduced by Picard and Queyranne [22], which is used for enumerating all minimum s-t cuts of a digraph (with arbitrary positive capacities). In our case, we consider the unit capacity case, and use this representation to construct a DAG representing the Birkhoff representation $(\Pi^*(\mathcal{R}_{st}), \preceq)$ of the distributive lattice $(\mathcal{R}_{st}, \subseteq)$, as well as the partition $\Pi(\mathcal{R}_{st})$ of V.

To begin, we briefly describe the representation proposed by Picard and Queyranne. Let f be an arbitrary maximum s-t flow of G, where each arc has unit capacity. Let G_f denote the residual graph of f. Then, the following characterization is known.

▶ **Lemma 12** ([22]). *An s-t cut $\Delta^+(X)$ is a minimum s-t cut in G if and only if X is a closed set under reachability in G_f, containing s but not t.*

Using this, Picard and Queyranne constructed a DAG $D = (V', A')$ from G through the following steps: (1) Contract each strongly connected component in G_f into a single vertex, (2) remove the component containing s along with all vertices reachable from s, and (3) remove the component containing t along with all vertices reachable to t. For each vertex $v' \in V'$, let $R(v')$ denote the set of vertices in G_f contracted into v'. Similarly, let $R(s')$ and $R(t')$ denote the sets of vertices in G_f removed in steps (2) and (3), respectively. By Lemma 12, the DAG D represents all minimum s-t cuts in G. Specifically, for any $X \subseteq V'$ that is closed under reachability in D, the set of outgoing arcs from $R(X) \coloneqq (\bigcup_{v' \in X} R(v')) \cup R(s')$ in G forms a minimum s-t cut in G. Conversely, for any minimum s-t cut $\Delta^+(X)$ in G, the set of components corresponding to $X \setminus R(s')$ is closed under reachability in D.

In fact, D is the desired DAG that represents the Birkhoff representation $(\Pi^*(\mathcal{R}_{st}), \preceq)$ of $(\mathcal{R}_{st}, \subseteq)$. Let $(V', \preceq)$ be a poset where reachability in D defines the partial order i.e., for any $u, v \in V'$, $u \preceq v$ if and only if u is reachable from v in D. From the above discussion, the map $R\colon \mathcal{I}(V') \to \mathcal{R}_{st}$ is an isomorphism between $(\mathcal{I}(V'), \subseteq)$ and $(\mathcal{R}_{st}, \subseteq)$, because for all $X, X' \in \mathcal{I}(V')$, we have $X \subseteq X'$ if and only if $R(X) \subseteq R(X')$. Moreover, choose an arbitrary maximal chain $\emptyset = X_0 \subsetneq X_1 \subsetneq \cdots \subsetneq X_n = V'$ in $\mathcal{I}(V')$. Since $R(X_0) \subsetneq R(X_1) \subsetneq \cdots \subsetneq R(X_n)$ forms a maximal chain from $X_\bot$ to $X_\top$ in $\mathcal{R}_{st}$, and $|X_i \setminus X_{i-1}| = 1$ for all $i \in [n]$, we obtain an isomorphism that maps each $v \in V'$ to $R(v)$, implying the equivalence between $(V', \preceq)$ and $(\Pi^*(\mathcal{R}_{st}), \preceq)$.

Thus, as all strongly connected components in G can be found in linear time by [24], the partition $\Pi(\mathcal{R}_{st})$ and a DAG representing $(\Pi^*(\mathcal{R}_{st}), \preceq)$ can be constructed from G_f in $O(|V| + |A|)$ time. Furthermore, the number of vertices and arcs representing $(\Pi^*(\mathcal{R}_{st}), \preceq)$ are at most $|V|$ and $|A|$, respectively.

We then define a map $\hat{r}\colon A \to V^2$ by

$$\hat{r}(a) \coloneqq \begin{cases} a & \text{if } f(a) = 1, \\ (t, t) & \text{if } f(a) = 0. \end{cases} \tag{1}$$

We confirm that $\hat{r}$ is a pre-reduction map.

▶ **Lemma 13** (⋆). *The map $\hat{r}\colon A \to V^2$ defined as* (1) *is a pre-reduction map.*

This together with Proposition 10 and the above argument implies that $T_P(G) + T_r(G) = O(|V| + |A|)$. Thus, by Theorem 8, we obtain the following, which implies Theorem 3 (1).

▶ **Theorem 14** (⋆). *The k-diverse problem with respect to d_φ of* Unweighted Minimum s-t Cut *can be solved in $O(n + (|B_k(\varphi)|m)^{1+o(1)} \log(\varphi(k)) \log k + kq)$ time, where n denotes the number of vertices in the input digraph G, m the number of arcs, and q the size of any minimum s-t cut in G. In particular,* Sum-k-Diverse *and* Cov-k-Diverse *of* Unweighted Minimum s-t Cut *can be solved in $O(n + (km)^{1+o(1)})$ time and in $O(n + m^{1+o(1)} \log^2 k + kq)$ time, respectively.*

5.3 The k-diverse stable matching problem

In this subsection, we consider the k-diverse problem of Stable Matching. Let us first introduce the problem Stable Matching (see e.g., [14] for details). Let U and V be disjoint finite sets with the same size n endowed with total orders $\leq_u$ on V ($u \in U$) and $\leq_v$ on U ($v \in V$). Intuitively, the total order $\leq_u$ (resp. $\leq_v$) represents the preference of u on V (resp. v

on U); $v <_u v'$ means that "u prefer v to v'." A subset $M = \{(u_1, v_1), (u_2, v_2), \ldots, (u_n, v_n)\} \subseteq U \times V$ is called a *matching* if M provides the one-to-one correspondence between U and V, i.e., all u_i, v_j are different, $U = \{u_1, u_2, \ldots, u_n\}$, and $V = \{v_1, v_2, \ldots, v_n\}$. For a matching M, we denote by $p_M(u)$ (resp. $p_M(v)$) the partner of $u \in U$ (resp. $v \in V$) in M, i.e., $(u, p_M(u)) \in M$ (resp. $(p_M(v), v) \in M$). A matching M is said to be *stable* if there is no pair $(u, v) \in U \times V$ such that $v <_u p_M(u)$ and $u <_v p_M(v)$. In Stable Matching, we are given a tuple $(U, V; (\leq_u)_{u \in U}, (\leq_v)_{v \in V})$ of finite sets U, V with the same size n and the total orders $(\leq_u)_{u \in U}$ on V and $(\leq_v)_{v \in V}$ on U, and asked to find a stable matching M. A stable matching always exists and can be found in $O(n^2)$ time by the *Gale–Shapley algorithm* [13].

Clearly, Stable Matching has the property (S), in which $q = |U| = |V|$. Similarly to Section 5.2, our aim is to construct, for some ring family $\mathcal{R}$, a DAG representing the Birkhoff representation $(\Pi^*(\mathcal{R}), \preceq)$, the partition $\Pi(\mathcal{R})$, and a pre-reduction map $\hat{r}$ based on $\mathcal{R}$.

Let $\mathbf{I} \coloneqq (U, V; (\leq_u)_{u \in U}, (\leq_v)_{v \in V})$ be an instance of Stable Matching, and let $n \coloneqq |U| = |V|$. Then, the family $\mathcal{S}(\mathbf{I})$ of solutions of the instance $\mathbf{I}$ is the set of all stable matchings of $\mathbf{I}$. It is known [14, Theorem 1.3.2] that $\mathcal{S}(\mathbf{I})$ forms a distributive lattice with the partial order $\preceq$ defined by setting $M \preceq M'$ if and only if $p_M(u) \leq_u p_{M'}(u)$ for all $u \in U$. For each stable matching $M \in \mathcal{S}(\mathbf{I})$, the *P-set* of M, denoted by $P(M)$, is the set of pairs (u, v) such that v is at least as preferred as $p_M(u)$ by $u \in U$, i.e., $P(M) \coloneqq \{(u, v) \in U \times V \mid v \leq_u p_M(u)\}$. Let $\mathcal{R}_{\mathbf{I}}$ denote the family of all P-sets for $\mathbf{I}$, i.e.,

$$\mathcal{R}_{\mathbf{I}} \coloneqq \{P(M) \mid M \in \mathcal{S}(\mathbf{I})\}.$$

The map that sends a stable matching to its P-set is an isomorphism between $(S(\mathbf{I}), \preceq)$ and $(\mathcal{R}_{\mathbf{I}}, \subseteq)$. Hence, $\mathcal{R}_{\mathbf{I}}$ forms a ring family, which serves the base of our pre-reduction map. We note that the unique minimal set $X_\perp$ of $\mathcal{R}_{\mathbf{I}}$ is nonempty. To ensure that $X_\top$ is a proper subset of the ground set, we add a new element $\top$ to V, and we regard $U \times (V \cup \{\top\})$ as the ground set in the following.

We can construct the Birkhoff representation $(\Pi^*(\mathcal{R}_{\mathbf{I}}), \preceq)$ of the distributive lattice $(\mathcal{R}_{\mathbf{I}}, \subseteq)$ following the work of Gusfield and Irving [14, Chapter 3]. For each stable matching M and each $u \in U$, let $s_M(u)$ denote the most preferable element $v \in V$ for u such that $u <_v p_M(v)$, if such a v exists. A rotation is an ordered list $\rho = ((u_0, v_0), (u_1, v_1), \ldots, (u_{c-1}, v_{c-1}))$ of pairs in some stable matching M, satisfying $s_M(u_i) = v_{i+1}$ for all $i \in [0, c-1]$ (where $i+1$ is taken modulo c). Let $\Lambda_{\mathbf{I}}$ be the set of all rotations of $\mathbf{I}$. It is shown that the map d defined by

$$d(\rho) = \{(u_i, v) \in U \times V \mid i \in [0, c-1], v_i <_{u_i} v \leq_{u_i} v_{i+1}\}$$

is a bijection between $\Lambda_{\mathbf{I}}$ and $\Pi^*(\mathcal{R}_{\mathbf{I}})$. A partial order on $\Lambda_{\mathbf{I}}$ is induced by $\Pi^*(\mathcal{R}_{\mathbf{I}})$ through d. Gusfield and Irving present an $O(n^2)$-time algorithm that constructs $\Lambda_{\mathbf{I}}$ along with a DAG representing the poset over $\Lambda_{\mathbf{I}}$ having $O(n^2)$ arcs [14, Lemma 3.3.2 and Corollary 3.3.1]. In particular, their algorithm computes the unique minimal and maximal stable matchings.

We can construct $\Pi^*(\mathcal{R}_{\mathbf{I}})$, which is a partition of $X_\top \setminus X_\perp$, from $\Lambda_{\mathbf{I}}$ in $O(n^2)$ time using d. Clearly, $X_\perp$ is the P-set of the unique minimal stable matching. Thus, we can construct the partition $\Pi(\mathcal{R}_{\mathbf{I}})$ and a DAG with $O(n^2)$ arcs representing $(\Pi^*(\mathcal{R}_{\mathbf{I}}), \preceq)$ in $O(n^2)$ time.

We then construct a pre-reduction map. For each $u \in U$ and $v \in V$, we denote by v_u^- the element in V that is the cover of v with respect to $\leq_u$, i.e., $v <_u v_u^-$ and there is no element v' in V with $v <_u v' <_u v_u^-$. If v is the maximum element with respect to $\leq_u$, then we define $v_u^- \coloneqq \top$. We define a map $\hat{r} \colon U \times V \to (U \times (V \cup \{\top\}))^2$ by

$$\hat{r}(u, v) \coloneqq \big((u, v), (u, v_u^-)\big). \tag{2}$$

We show that this is a pre-reduction map.

▶ **Lemma 15** (⋆). *The map* $\hat{r}\colon U \times V \to ((U \times V) \cup \{\top\})^2$ *defined as* (2) *is a pre-reduction map.*

This together with Proposition 10 and the above argument implies that we have $T_P(\mathbf{I}) + T_r(\mathbf{I}) = O(n^2)$. Thus, by Theorem 8, we obtain the following, which implies Theorem 3 (2).

▶ **Theorem 16** (⋆). *The* k*-diverse problem with respect to* d_φ *of* Stable Matching *can be solved in* $O((|B_k(\varphi)|n^2)^{1+o(1)} \log(\varphi(k)) \log k + kn)$ *time. In particular,* Sum-k-Diverse *of* Stable Matching *and* Cov-k-Diverse *of* Stable Matching *can be solved in* $O((kn^2)^{1+o(1)})$ *time and in* $O(n^{2+o(1)} \log^2 k + kn)$ *time, respectively.*

5.4 The k-diverse problem on the product of total orders

Very recently, De Berg, Martínez, and Spieksma [8] introduce a framework for the k-diverse problem on the product of total orders, which leads to the polynomial-time solvability of Sum-k-Diverse/Cov-k-Diverse of Unweighted Minimum s-t Cut/Stable Matching. In this subsection, we briefly introduce their framework and show that our framework can capture theirs for Sum-k-Diverse and Cov-k-Diverse. Here, we do *not* assume that Prob has the properties (S) and (R).

We first introduce some terminology. For a distributive lattice $(\mathcal{L}, \preceq)$, an element $x \in \mathcal{L}$ is said to be *join-irreducible* if $x \neq y \vee z$ for any $y, z \in \mathcal{L} \setminus \{x\}$. Let $\mathcal{L}_{\mathrm{ir}}$ denote the set of join-irreducible elements in $\mathcal{L}$. Then, it is known [5] that the subposet $(\mathcal{L}_{\mathrm{ir}}, \preceq)$ of $\mathcal{L}$ induced by $\mathcal{L}_{\mathrm{ir}}$ forms a Birkhoff representation of $\mathcal{L}$; the map $I \mapsto \bigvee_{x \in I} x$ is an isomorphism from $(\mathcal{I}(\mathcal{L}_{\mathrm{ir}}), \subseteq)$ to $(\mathcal{L}, \preceq)$. We refer to this Birkhoff representation $(\mathcal{L}_{\mathrm{ir}}, \preceq)$ as the *join-irreducible representation* of $\mathcal{L}$. For total orders $(E_1, \leq_1), (E_2, \leq_2), \dots, (E_q, \leq_q)$, their *product* $(\mathcal{E}, \leq)$ is the poset such that its ground set $\mathcal{E}$ is the product $E_1 \times E_2 \times \cdots \times E_q$ of $E_1, E_2, \dots, E_q$ and the partial order $\leq$ is defined by setting $(e_1, e_2, \dots, e_q) \leq (e'_1, e'_2, \dots, e'_q)$ if and only if $e_i \leq_i e'_i$ for all $i \in [n]$. Actually, this $\mathcal{E}$ forms a lattice; the meet $(e_1, e_2, \dots, e_q) \wedge (e'_1, e'_2, \dots, e'_q)$ is given by $(\min\{e_i, e'_i\})_{i \in [q]}$ and the join $(e_1, e_2, \dots, e_q) \vee (e'_1, e'_2, \dots, e'_q)$ is given by $(\max\{e_i, e'_i\})_{i \in [q]}$, where

$$\min\{e_i, e'_i\} \coloneqq \begin{cases} e_i & \text{if } e_i \leq_i e'_i, \\ e'_i & \text{if } e'_i <_i e_i, \end{cases} \qquad \max\{e_i, e'_i\} \coloneqq \begin{cases} e'_i & \text{if } e_i \leq_i e'_i, \\ e_i & \text{if } e'_i <_i e_i \end{cases}$$

for each $i \in [n]$. We say that a subset $\mathcal{L} \subseteq \mathcal{E}$ is a *sublattice* if $\mathcal{L}$ is closed under the meet $\wedge$ and join $\vee$, i.e., $x, y \in \mathcal{L}$ implies $x \wedge y, x \vee y \in \mathcal{L}$. We can observe that a sublattice of the product of total orders is distributive (see the paragraph after Theorem 17).

De Berg, Martínez, and Spieksma [8] impose the following property on Prob:

(T) For any instance $\mathbf{I}$ of Prob, there are q total orders $(E_1, \leq_1), (E_2, \leq_2), \dots, (E_q, \leq_q)$ such that the family $\mathcal{S}(\mathbf{I})$ of solutions of $\mathbf{I}$ is a sublattice of the product of those total orders.

Then they show that, if Prob has the property (T) and we can construct (a DAG representing) the join-irreducible representation $(\mathcal{S}(\mathbf{I})_{\mathrm{ir}}, \leq)$ of $\mathcal{S}(\mathbf{I})$ in polynomial time, then we can solve Sum-k-Diverse/Cov-k-Diverse of Prob in polynomial time by using an algorithm for the SFM problem over the distributive lattice $\mathcal{I}(\mathcal{S}(\mathbf{I})_{\mathrm{ir}})$. The problem Stable Matching has the property (T). Indeed, for an instance $\mathbf{I} \coloneqq (U, V; (\leq_u)_{u \in U}, (\leq_v)_{v \in V})$ of Stable Matching, we define $E_u \coloneqq \{(u, v) \mid v \in V\}$ for each $u \in U$ and extend the total order $\leq_u$ on V to that on E_u by setting $(u, v) \leq_u (u, v')$ if and only if $v \leq_u v'$. Then, $\mathcal{S}(\mathbf{I})$ forms a sublattice of the product of $(E_u, \leq_u)$ for all $u \in U$. Similarly, Unweighted Minimum s-t Cut also has the property (T) by introducing the *left-right order* to q arc disjoint paths, where q denotes the size of a minimum s-t cut (or the maximum number of arc disjoint s-t paths); see [7, 8] for details.

We can show the following, whose proof is almost the same as that in Section 5.3.

▶ **Theorem 17** (⋆). *If a combinatorial problem* Prob *has the property (T), then it also has the properties (S) and (R).*

Here, for a solution $X = (e_1, e_2, \dots, e_q) \in \mathcal{S}(\mathbf{I})$, we define its *P-set* $P(X)$ by $P(X) \coloneqq \{(e'_1, e'_2, \dots, e'_q) \mid e'_i \leq_i e_i \ (i \in [q])\}$ and denote by $\mathcal{R}_{\mathbf{I}}$ the family of all P-sets of $\mathbf{I}$, as in Section 5.3. Then we can easily see that $(\mathcal{S}(\mathbf{I}), \leq)$ and $(\mathcal{R}_{\mathbf{I}}, \subseteq)$ are isomorphic, both of which are distributive.

Let us see the result of De Berg, Martínez, and Spieksma [8] in detail to compare the time complexity of their algorithm with ours. Let E be the disjoint union (or direct sum) of $E_1, E_2, \dots, E_q$. We denote by $T_{\mathrm{ir}}(\mathbf{I})$ the time required to construct a DAG representing the join-irreducible representation $(\mathcal{S}(\mathbf{I})_{\mathrm{ir}}, \leq)$, and by $T_{\mathrm{SFM}}(n, \mathrm{EO})$ the time required to minimize a submodular function f with n variables such that one value evaluation of f takes EO time. Then, it is shown in [8] that we can solve Sum-k-Diverse/Cov-k-Diverse of Prob having the property (T) in $O(T_{\mathrm{ir}}(\mathbf{I}) + T_{\mathrm{SFM}}(k|E|, k^2|E|q))$ time. Using the state-of-the-art algorithm for SFM given in [18] with we have $T_{\mathrm{SFM}}(n, \mathrm{EO}) = O(n^3\mathrm{EO})$. Hence, the running-time of the algorithms of De Berg, Martínez, and Spieksma is $O(T_{\mathrm{ir}}(\mathbf{I}) + k^5|E|^4 q)$.

Our framework provides much faster algorithms for Sum-k-Diverse and Cov-k-Diverse of Prob as follows.

▶ **Theorem 18** (⋆). *Suppose that a combinatorial problem* Prob *has the property (T). Then we can solve* Sum-k-Diverse *and* Cov-k-Diverse *of* Prob *in* $O(T_{\mathrm{ir}}(\mathbf{I}) + |E|q + (|A| + k|E|)^{1+o(1)})$ *time and in* $O(T_{\mathrm{ir}}(\mathbf{I}) + (|E| + k)q + (|A| + |E|)^{1+o(1)} \log^2 k)$ *time for an instance* $\mathbf{I}$ *of* Prob*, respectively, where A denotes the arc set of a constructed DAG representing* $(\mathcal{S}(\mathbf{I})_{\mathrm{ir}}, \leq)$.

Since $|\mathcal{S}(\mathbf{I})_{\mathrm{ir}}|$ is upper-bounded by the length of a maximal chain in the product of total orders $(E_1, \leq_1), (E_2, \leq_2), \dots, (E_q, \leq_q)$, we have $|\mathcal{S}(\mathbf{I})_{\mathrm{ir}}| = O(|E|)$. Thus, $|A|$ is upper-bounded by $|\mathcal{S}(\mathbf{I})_{\mathrm{ir}}|(|\mathcal{S}(\mathbf{I})_{\mathrm{ir}}| - 1) = O(|E|^2)$. Even in the worst case of $|A| = \Theta(|E|^2)$, the running-time of our algorithms for Sum-k-Diverse and Cov-k-Diverse of Prob are $O(T_{\mathrm{ir}}(\mathbf{I}) + (|E|^2 + k|E|)^{1+o(1)})$ time and $O(T_{\mathrm{ir}}(\mathbf{I}) + kq + |E|^{2+o(1)} \log^2 k)$ time, respectively, which are much faster than the previous ones that take $O(T_{\mathrm{ir}}(\mathbf{I}) + k^5|E|^4 q)$ time.

We also mention that the framework of [8] works for a diversity measure based on lengths of the maximal chains. It may be interesting to generalize our result to treat such a measure.

References

1. R. K. Ahuja, D. S. Hochbaum, and J. B. Orlin. Solving the convex cost integer dual network flow problem. *Management Science*, 49(7):950–964, 2003. doi:10.1287/MNSC.49.7.950.16384.
2. F. Bach. Learning with submodular functions: A convex optimization perspective. *Foundations and Trends® in Machine Learning*, 6(2-3):145–373, 2013. doi:10.1561/2200000039.
3. J. Baste, M. R. Fellows, L. Jaffke, T. Masařík, M. de Oliveira Oliveira, G. Philip, and F. A. Rosamond. Diversity of solutions: An exploration through the lens of fixed-parameter tractability theory. *Artificial Intelligence*, 303(103644):1–15, 2022.
4. J. Baste, L. Jaffke, T. Masařík, G. Philip, and G. Rote. FPT algorithms for diverse collections of hitting sets. *Algorithms*, 12(12):254, 2019. doi:10.3390/A12120254.
5. G. Birkhoff. Rings of sets. *Duke Mathematical Journal*, 3(3):443–454, 1937.
6. B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 2nd edition, 2002.
7. M. de Berg, A. L. Martínez, and F. Spieksma. Finding diverse minimum s-t cuts. In *Proceedings of the 34th International Symposium on Algorithms and Computation (ISAAC 2023)*, volume 283, pages 24:1–24:17, 2023. doi:10.4230/LIPICS.ISAAC.2023.24.

8 M. de Berg, A. L. Martínez, and F. Spieksma. Finding diverse solutions in combinatorial problems with a distributive lattice structure. arXiv:2504.02369, 2025. doi:10.48550/arXiv.2504.02369.

9 E. Eiben, T. Koana, and M. Wahlström. Determinantal sieving. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2024)*, pages 377–423, 2024.

10 F. V. Fomin, P. A. Golovach, L. Jaffke, G. Philip, and D. Sagunov. Diverse pairs of matchings. *Algorithmica*, 86(6):2026–2040, 2024. doi:10.1007/S00453-024-01214-7.

11 F. V. Fomin, P. A. Golovach, F. Panolan, G. Philip, and S. Saurabh. Diverse collections in matroids and graphs. *Mathematical Programming*, 204(1):415–447, 2024. doi:10.1007/S10107-023-01959-Z.

12 S. Fujishige. *Submodular Functions and Optimization*. Annals of Discrete Mathematics. Elsevier Science, London, England, 2nd edition, 2005.

13 D. Gale and L. S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.

14 D. Gusfield and R. W. Irving. *The Stable Marriage Problem: Structure and Algorithms*. Foundations of Computing Series. The MIT Press, 1989.

15 T. Hanaka, M. Kiyomi, Y. Kobayashi, Y. Kobayashi, K. Kurita, and Y. Otachi. A framework to design approximation algorithms for finding diverse solutions in combinatorial problems. In *Proceedings of the 37th AAAI Conference on Artificial Intelligence (AAAI 2023)*, volume 37, pages 3968–3976, 2023. doi:10.1609/AAAI.V37I4.25511.

16 T. Hanaka, Y. Kobayashi, K. Kurita, S. W. Lee, and Y. Otachi. Computing diverse shortest paths efficiently: A theoretical and experimental study. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence (AAAI 2022)*, volume 36, pages 3758–3766, 2022. doi:10.1609/AAAI.V36I4.20290.

17 T. Hanaka, Y. Kobayashi, K. Kurita, and Y. Otachi. Finding diverse trees, paths, and more. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI 2021)*, volume 35, pages 3778–3786, 2021. doi:10.1609/AAAI.V35I5.16495.

18 H. Jiang. Minimizing convex functions with rational minimizers. *Journal of the ACM*, 70(1):5:1–5:27, 2022. doi:10.1145/3566050.

19 B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer, Berlin, Heidelberg, 6th edition, 2018.

20 K. Murota. *Discrete Convex Analysis*. Society for Industrial and Applied Mathematics, 2003.

21 K. Murota. *Matrices and Matroids for Systems Analysis*. Springer, Berlin, Heidelberg, 2010.

22 J.-C. Picard and M. Queyranne. *On the structure of all minimum cuts in a network and applications*, volume 13 of *Mathematical Programming Studies*, pages 8–16. Springer, 1980.

23 A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Algorithms and Combinatorics. Springer, 2003.

24 M. Sharir. A strong-connectivity algorithm and its applications in data flow analysis. *Computers & Mathematics with Applications*, 7(1):67–72, 1981.

25 J. van den Brand, L. Chen, R. Kyng, Y. P. Liu, R. Peng, M. P. Gutenberg, S. Sachdeva, and A. Sidford. A deterministic almost-linear time algorithm for minimum-cost flow. In *Proceedings of the IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS 2023)*, pages 503–514, 2023. doi:10.1109/FOCS57990.2023.00037.

Parameterized Reunion with Achromatic Number

Satyabrata Jana
University of Warwick, UK

Souvik Saha
The Institute of Mathematical Sciences, HBNI, Chenai, India

Saket Saurabh
The Institute of Mathematical Sciences, HBNI, Chenai, India
University of Bergen, Norway

Anannya Upasana
The Institute of Mathematical Sciences, HBNI, Chenai, India

Abstract

In this paper, we study the Achromatic Number problem. Given a graph G and an integer k, the task is to determine whether there exists a proper coloring of G, using at least k colors, in which every pair of distinct colors appears on the endpoints of some edge. It was established early on that the problem is fixed-parameter tractable (FPT)– even before the formal development of parameterized complexity. In fact, Farber, Hahn, Hell, and Miller [JCTB, 1986] devised an algorithm with a running time of $\mathcal{O}(f(k) \cdot |E(G)|)$. Although the exact form of $f(k)$ was not specified, it appears to be at least doubly exponential in k. In our work, we first present an algorithm with an explicit dependence on k, and then introduce another algorithm that is parameterized by the vertex cover number of the graph. More formally, we show the following.

- Achromatic Number is solvable in time $2^{\mathcal{O}(k^5)} + \mathcal{O}(|E(G)|)$.
- Achromatic Number admits a polynomial kernel when the input is restricted to a d-degenerate graph and a more efficient kernel on trees.
- We also study the parameterized complexity of the problem with respect to Vertex Cover and show that it admits an FPT algorithm running in time $2^{\mathcal{O}(\ell^2)} \cdot n^{\mathcal{O}(1)}$, where ℓ is the size of a vertex cover.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases Achromatic number, Coloring, Fixed-parameter tractability, Kernelization, Lower bound, W-hardness

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.42

Funding *Satyabrata Jana*: Supported by the Engineering and Physical Sciences Research Council (EPSRC) via the project MULTIPROCESS (grant no. EP/V044621/1).
Saket Saurabh: Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 819416).

1 Introduction

In this paper, we revisit the Achromatic Number problem that had a fixed parameter tractable (FPT) algorithm even before the concept of parameterized complexity was established, as we currently understand it. A *proper coloring* of a graph is an assignment of colors to the vertices such that no two endpoints of an edge are assigned the same color. A *complete coloring* of a graph is a proper coloring of a graph such that for any pair of distinct colors, there exists a pair of adjacent vertices that have been assigned those colors. The *achromatic number* of a graph G, denoted by $\psi(G)$, is the maximum number of colors that can be used in a complete coloring of G. In this paper, we study the following problem.

© Satyabrata Jana, Souvik Saha, Saket Saurabh, and Anannya Upasana;
licensed under Creative Commons License CC-BY 4.0

ACHROMATIC NUMBER **Parameter:** k
Input: A graph G on n vertices and m edges and an integer k.
Question: Is $\psi(G) \geq k$?

The computational complexity of ACHROMATIC NUMBER was first considered by Yannakakis and Gavril in 1980, who showed that the problem is NP-complete [22, Corollary 2]. The authors also observed that, unlike the classical CHROMATIC NUMBER problem, which is known to be NP-complete even for $k = 3$ (that is, whether the input graph can be properly colored with 3 colors), ACHROMATIC NUMBER cannot be shown to be NP-complete for a fixed value of k. Toward this, they show that for a graph G, $\psi(G) \geq k$ if and only if there is a subset $W \subseteq V(G)$ of size at most $\binom{k}{2}$ such that $\psi(G[W]) \geq k$. This immediately leads to an algorithm with running time $\mathcal{O}(n^{k^2})$. In the modern terminology of parameterized complexity (PC), this is an XP algorithm for ACHROMATIC NUMBER. Thus, a natural question is whether there exists an algorithm with running time $f(k)n^{\mathcal{O}(1)}$, that is, whether the problem admits an FPT algorithm.

A natural question is whether there exists a uniform polynomial-time algorithm for every fixed k. That is, do we have $n^{\mathcal{O}(1)}$ time algorithm where $\mathcal{O}(1)$ does not depend on k. (Pre PC Era)
In modern terminology, does there exist an algorithm with running time $f(k) \cdot n^{\mathcal{O}(1)}$, that is, does the problem admit an FPT algorithm. (Post PC Era)

In 1986, Farber, Hahn, Hell and Miller [10] designed a "linear time algorithm" for ACHROMATIC NUMBER. That is, an algorithm with running time $\mathcal{O}(m)$; which is actually an algorithm with running time $f(k) \cdot m$. This established the parameterized complexity of ACHROMATIC NUMBER. The exact value of $f(k)$ in the running time is not given and it appears to be at least doubly exponential in k. This algorithm was built on an earlier work of Hell and Miller [14] done in 1976 who showed that the number of irreducible graphs (without any twins) with $\psi(G) = k$ is upper bounded by some function $h(k)$. These results are the starting point of our research. Our aim is to design an algorithm with explicit dependence on k by a function that grows as slowly as possible. Furthermore, we also explore the problem with a structural parameter.

Related Works. Bounds on ACHROMATIC NUMBER in terms of other graph variants like vertex cover number and independence number have been studied [5, 12]. The PSEUDO-ACHROMATIC NUMBER problem, a related problem, was introduced in 1969 [13] and has been studied extensively [3, 4, 21]. The PSEUDO-ACHROMATIC NUMBER problem or GRAPH COMPLETE PARTITION problem checks whether an undirected graph can be partitioned into k classes such that every pair of classes is connected by an edge. Unlike the achromatic number problem, we do not require these classes to be independent sets. The pseudo-achromatic number may strictly exceed the achromatic Number even for a family of trees [9]. Halldórsson et al studied the problem from the approximation viewpoint, giving tight lower and upper bounds on its approximability. The problem has a $\mathcal{O}(k^2)$ kernel that also establishes that it is FPT parameterized by the solution size [7]. Another related problem is the HARMONIOUS COLORING where the objective is to find a proper coloring of the graph such that each pair of colors appears together on at most one edge [15]. It is known to be NP-hard on trees, interval and permutation graphs [8, 2, 20]. Georges investigated the problem on paths, cycles, complete graphs and complete bipartite graphs [11]. Kolay et al [17] studied it from an

FPT perspective and showed that it is FPT parameterized by the solution size as well as the vertex cover number of the graph. They also gave an exact exponential time algorithm in split graphs.

1.1 Our Results and Methods with Solution Size as a Parameter

Our first main result is the following FPT algorithm for Achromatic Number.

▶ **Theorem 1.** Achromatic Number *can be solved in* $2^{\mathcal{O}(k^5)} + \mathcal{O}(m)$ *time.*

The idea behind the proof of Theorem 1 is motivated by results in the area of approximation algorithms for an optimization variant of Achromatic Number. Kortsarz and Krauthgamer gave an algorithm that approximates the achromatic number within a ratio of $\mathcal{O}\left(\frac{n \cdot \log\log n}{\log n}\right)$ for general graphs [18]. This algorithm in turn was motivated by an earlier combinatorial work of Máté [19] on Achromatic Number on irreducible graphs. Our parameterized algorithm builds on this approximation algorithm and incorporates new ideas and concepts in several places to transform it into an FPT algorithm. We also mention in passing that Kortsarz and Krauthgamer [18] showed that there is no $(2-\epsilon)$-approximation algorithm for every fixed $\epsilon > 0$, unless P=NP.

The next natural question is whether the problem admits a polynomial kernel. That is, in polynomial time, could we replace the given instance by an equivalent instance of size polynomial in k? We do not know the answer to this question, and we leave this as a challenging open problem. However, could we say something when the input instances are restricted to some structured graph classes? In this regard, we first mention that Achromatic Number is known to be NP-complete even when the input graph is a tree [6]. Thus, researchers have considered Achromatic Number on several classes of trees [10] and designed polynomial time algorithms on these graph classes. In this paper, we design polynomial kernels for Achromatic Number on trees and d-degenerate graphs and obtain the following results.

▶ **Theorem 2 (♠).** Achromatic Number *admits a kernel of size* $\mathcal{O}(k^2)$ *on forests.* [1]

▶ **Theorem 3.** Achromatic Number *admits a kernel of size* $\mathcal{O}(k^{24d+2})$ *on* d-degenerate *graphs.*

1.2 Our Results and Methods for Structural Parameterization

We further study the Achromatic Number problem with respect to other parameters. We define the problem Achromatic Number/$\texttt{Mod}(\pi)$, where π is a graph class, as follows.

Achromatic Number/$\texttt{Mod}(\pi)$ **Parameter:** ℓ
Input: An undirected graph G, a modulator of size ℓ to π and a positive integer k.
Question: Is $\psi(G) \geq k$?

Recall that Achromatic Number is known to be NP-complete even when the input graph is a tree [6]. This immediately rules out the treewidth or the size of the feedback vertex set as parameters.

[1] Due to space constraints, the proofs of results marked with ♠ have been omitted and will appear in the full version of the paper.

Modulator to edgeless graphs. Our parameter of study is the vertex cover number of the graph, that is, a vertex set whose deletion results in an edgeless graph. Here, π is the family of edgeless graphs. Our result in this direction is the following.

▶ **Theorem 4.** Achromatic Number/VC *can be solved in* $2^{\mathcal{O}(\ell^2)} \cdot n^{\mathcal{O}(1)}$ *time.*

To prove Theorem 4, we first show that the achromatic number of the graph is bounded by $\ell + 1$. Then, we guess the color of the vertices in the modulator (one that will result in a solution). It is possible that there exists a pair of colors that is not assigned to the endpoints of any edge, for e.g., there may not be an edge between color classes C_1 and C_2. To address this issue, we utilize a vertex v outside the modulator and assign it either color C_1 or C_2, thereby *resolving the pair* formed by color classes C_1 and C_2 (in essence, we resolve the conflict between color classes C_1 and C_2 that arises because there is no edge between the color classes). We solve this by reducing the problem to an instance of Subgraph Isomorphism (given a host graph G and a pattern graph H, find a copy of H in G), where the pattern graph H has size $\mathcal{O}(\ell^2)$ and treewidth $\mathcal{O}(1)$. It is known that Subgraph Isomorphism can be solved in time $2^{\mathcal{O}(|H|)} \cdot |V(G)|^{tw+1}$, where tw is the treewidth of the pattern graph, leading to the desired running time of our algorithm.

2 Preliminaries

In this study, we examine undirected graphs. For a given graph G, we represent its set of vertices as $V(G)$ and its set of edges as $E(G)$. When the context allows, we will refer to these simply as V and E. n refers to the number of vertices and m represents the number of edges. For any vertex $u \in V$, the set of vertices adjacent to u in G is represented by $N_G(u)$. In particular, $N_G(u)$ denotes the open neighborhood of u in G. We will abbreviate this to $N(u)$ when the graph is clear from the context. The degree of a vertex u, denoted as $deg(u)$, refers to the number of vertices in its neighborhood, i.e., $deg(u) = |N(u)|$. We call a graph *irreducible* or *twin-free* if any pair of distinct vertices has distinct open neighborhoods. A graph is called d-degenerate if every subgraph has a vertex with degree at most d. An induced matching or an independent matching is a set of edges such that no two endpoints of distinct edges of the matching are adjacent to each other. A *semi-independent matching* M is a set of edges $\{(x_1, y_1), (x_2, y_2), \ldots, (x_{|M|}, y_{|M|})\}$ such that the sets $X = \{x_1, x_2, \ldots, x_{|M|}\}$ and $Y = \{y_1, y_2, \ldots, y_{|M|}\}$ are independent and for any x_i and y_j with $i < j$, x_i is not adjacent to y_j (see Figure 1). We call a matching M maximal if, after adding any edge in $E(G) \backslash M$ to M, M does not remain a matching. An independent set is a set of vertices that are pairwise non-adjacent. An independent set $S \subseteq V(G)$ is called maximal if adding any vertex from $V(G) \backslash S$ to S destroys its property of being an independent set. For a given proper coloring of a graph, a color class refers to a set of vertices that are assigned the same color. A partial complete coloring of G is a complete coloring of a subset of vertices of G. A partial complete k-coloring is a partial complete coloring with k colors. A *greedy independent partition* of a graph G is an ordered partition of the vertex set $V(G)$ into independent sets, constructed through a greedy process, prioritizing the largest set first while ignoring the order of the rest. At each step, a maximal independent set is selected from the remaining vertices and the process continues until all vertices are covered. The size of a *greedy independent partition* is the number of independent sets in the partition.

3 FPT algorithm for Achromatic Number

In this section, we give an FPT algorithm for Achromatic Number parameterized by solution size in general graphs. We start with describing our kernelization procedure that takes an instance (G, k) of Achromatic Number as input and in $\mathcal{O}(|E(G)|)$ time return an equivalent instance (kernel). We prove the following result.

▶ **Theorem 5.** Achromatic Number *admits a kernel of size* $k^{k+2}2^{k^3+k+2}e^{k^2}$. *And moreover we can find it in* $\mathcal{O}(m)$ *time where* m *is the number of edges in the given graph.*

Our kernelization procedure for a given instance (G, k) involves the following steps. Let R be an equivalence relation defined as follows: for a vertex $v \in V(G)$, the class R_v contains the vertices $\{u \mid N_G(u) = N_G(v)\}$.

1. If $n \leq k^{k+2}2^{k^3+k+2}e^{k^2}$, we return G as kernel.
2. Using Lemma 13, we check in $\mathcal{O}(|E(G)|)$ time whether the number of equivalence classes exceeds $k^{k+1}2^{k^3+k+2}e^{k^2}$, or else obtain their count q.
   a. If the number of equivalence classes exceeds $k^{k+1}2^{k^3+k+2}e^{k^2}$, return a trivial YES-instance of Achromatic Number.
   b. Otherwise, $q \leq k^{k+1}2^{k^3+k+2}e^{k^2}$. Apply Reduction Rules 1 and 2 exhaustively, and return the reduced instance as a kernel of size $k^{k+2}2^{k^3+k+2}e^{k^2}$.

We now show the correctness of this procedure. Toward that, we have to prove two things. First, we need to show that if the number of equivalence classes in G is more than $k^{k+1}2^{k^3+k+2}e^{k^2}$ then $\psi(G) \geq k$ (Lemma 15). And secondly, we have to show that after exhaustive application of the Reduction Rules 2 and 1 the size of $V(G)$ gets reduced to $k^{k+2}2^{k^3+k+2}e^{k^2}$ (Lemma 16). An example of a trivial YES instance of Achromatic Number is $(K_{1,1}, 1)$. We now describe the Reduction Rule 1 that removes all isolated vertices from the graph. The proof of the rule follows from the fact that removing isolated vertices from the graph will not affect its achromatic number, since no edge in the graph is incident to that vertex.

▶ **Reduction Rule 1.** *If* G *has an isolated vertex* v*, then return the instance* $(G - v, k)$.

Now mention two known facts regarding complete coloring of a graph.

▶ **Lemma 6** ([18, Lemma 1.1])**.** *A partial complete coloring in a graph* G *can be extended to a complete coloring of* G *in time* $\mathcal{O}(|E(G)|)$.

▶ **Lemma 7** ([18, Lemma 1.4])**.** *Given a semi-independent matching of a graph* G *of size at least* $\binom{k}{2}$*, a partial complete* k*-coloring can be computed in time* $\mathcal{O}(|E(G)|)$.

Lemma 6 and Lemma 7 together conclude the following lemma.

▶ **Lemma 8.** *If a graph* G *has a semi-independent matching of size* $\binom{k}{2}$*, then* $\psi(G) \geq k$.

We now prove the following crucial lemma.

▶ **Lemma 9.** *Let* G *be an irreducible graph with* $|V(G)| > k^{k+1}2^{k^3+k+2}e^{k^2}$. *Then* $\psi(G) \geq k$.

Proof. Let G be an irreducible graph. We show that if G has more than $k^{k+1}2^{k^3+k+2}e^{k^2}$ vertices then $\psi(G) \geq k$. Moreover, we design an algorithm for such a graph, i.e., we give a complete coloring with at least k colors. We first describe the algorithm briefly.

We start with finding a greedy independent partition Π. If the size of Π is at least k, we have that $\psi(G) \geq k$ (by Lemma 17). Moreover Π gives a complete coloring with at least k colors by assigning a distinct color to each set. Next, we focus on the first independent set S_0 in Π. At each recursive step i, we construct $S_{i+1} \subset S_i$ as follows:

- We fix an arbitrary vertex ordering.
- Let x_{2i} and x_{2i+1} be the two minimal vertices with respect to the ordering.
- We identify a *distinguishing vertex* d_{xy} that is adjacent to exactly one of x_{2i} and x_{2i+1}.
- Then we partition S_i into S_i^0 ($\subseteq N(d_{xy})$) and S_i^1 (not intersect $N(d_{xy})$).
- We set $S_{i+1} = S_i^0$ if $|S_i^0| \geq |S_i|/k$; otherwise, set $S_{i+1} = S_i^1$.

This process yields nested independent sets $S_0 \supset S_1 \supset \cdots \supset S_l$, forming either a *partial complete coloring* or a *semi-independent matching*, both can be extendable to a complete coloring. We also store a pair of sets U^0 and U^1 of witness vertices. Simultaneously we keep two index set Q^0 and Q^1. Next we show that for $|V(G)| \geq k^{k+1}2^{k^3+k+2}e^{k^2}$ the Algorithm 1 always terminates. In addition we show that either of the cases $Q^0 \geq k$ or $Q^1 \geq k^3$ leads to a complete coloring with at least k colors (Claims 11 and 12). A complete description of this above mentioned procedure is given in Algorithm 1. Now we show that Algorithm 1 always terminates. Assume that $p < k$ inside Algorithm 1 as otherwise it returns $\psi(G) \geq k$ so terminates. Note that if the Algorithm 1 does not terminate at Step 3 then the while loop (Step 11) runs at most $(k + k^3)$ steps as per our construction of the set Q_0 and Q_1. Now we show that either $|Q^0| \geq k$ or $|Q^1| \geq k^3$ in the output of Algorithm 1.

▷ Claim 10. Let G be an irreducible graph with more than $k^{k+1}2^{k^3+k+2}e^{k^2}$ vertices. If the Algorithm 1 outputs the pair of sets Q_0 and Q_1 then either $|Q^0| \geq k$ or $|Q^1| \geq k^3$.

Proof. We prove this by contradiction. Assume G is an irreducible graph with $n > k^{k+1}2^{k^3+k+2}e^{k^2}$ vertices and the Algorithm 1 outputs the pair of sets Q_0 and Q_1 with $|Q^0| < k$ or $|Q^1| < k^3$. Let $S_{\alpha,\beta}$ denote the independent set obtained after applying Step 14 α times and Step 15 β times, starting from S_0. We claim that

$$|S_{\alpha,\beta}| \geq \left(\frac{1}{2k}\right)^{\alpha} \left(\frac{1}{2}\left(1 - \frac{1}{k}\right)\right)^{\beta} |S_0| \tag{1}$$

Base case: When $\alpha + \beta = 0$, we have $S_{\alpha,\beta} = S_0$, so the inequality trivially holds.

Inductive step: By the construction of the algorithm:

$$|S_{\alpha,\beta}| \geq \min\left\{\frac{1}{2k}|S_{\alpha-1,\beta}|, \frac{1}{2}\left(1 - \frac{1}{k}\right)|S_{\alpha,\beta-1}|\right\}$$

By induction hypothesis:

$$|S_{\alpha-1,\beta}| \geq \left(\frac{1}{2k}\right)^{\alpha-1} \left(\frac{1}{2}\left(1 - \frac{1}{k}\right)\right)^{\beta} |S_0|, \quad |S_{\alpha,\beta-1}| \geq \left(\frac{1}{2k}\right)^{\alpha} \left(\frac{1}{2}\left(1 - \frac{1}{k}\right)\right)^{\beta-1} |S_0|$$

Thus, in either case, i.e., $|S_{\alpha,\beta}| \geq \frac{1}{2k}|S_{\alpha-1,\beta}|$ or $|S_{\alpha,\beta}| \geq \frac{1}{2}\left(1 - \frac{1}{k}\right)|S_{\alpha,\beta-1}|$ Equation (1) is satisfied. Now, since $|Q^0| < k$ and $|Q^1| < k^3$, the loop terminates after at most $k + k^3$ steps. Therefore, the set S_{k,k^3} has size at most 4, but also satisfies:

$$|S_{k,k^3}| \geq \left(\frac{1}{2k}\right)^{k} \left(\frac{1}{2}\left(1 - \frac{1}{k}\right)\right)^{k^3} |S_0|$$

Algorithm 1 Algorithm for the case when the graph is large and irreducible.

Input: An irreducible graph G with $n > k^{k+1}2^{k^3+k+2}e^{k^2}$ vertices.
Output: Either $\psi(G) \geq k$ or a pair of sets Q^0 and Q^1.

1: Compute a greedy independent partition $I_1, \ldots, I_p$ of G
2: **if** $p \geq k$ **then**
3: **return** $\psi(G) \geq k$
4: Let $I = I_\alpha$ where $|I_\alpha| \geq |I_j|, \forall j \in [p]$, with a total order $\prec$ on I ▷ $|I| \geq n/k$
5: **for all** $x, y \in I$ **do**
6: Choose d_{xy} adjacent to exactly one of x, y
7: $c(x, y) \leftarrow$ color of d_{xy}
8: **for all** $z \succ y \succ x$ in I **do**
9: $g(x, y, z) \leftarrow \begin{cases} 0 & \text{if } (z, d_{xy}) \in E \\ 1 & \text{otherwise} \end{cases}$
10: Initialize $S_0 = I$, $Q^0 = Q^1 = \emptyset$, $i = 0$
11: **while** $|S_i| > 4$ **do**
12: $x_{2i} \leftarrow \min_\prec S_i$, $x_{2i+1} \leftarrow$ next in S_i
13: Partition $S_i \setminus \{x_{2i}, x_{2i+1}\}$ into S_i^0 and S_i^1:

$$S_i^0 = \{x \mid g(x_{2i}, x_{2i+1}, x) = 0\}, S_i^1 = \{x \mid g(x_{2i}, x_{2i+1}, x) = 1\}$$

14: **if** $|S_i^0| \geq \frac{|S_i|}{2k}$ **then** $S_{i+1} \leftarrow S_i^0$, $Q^0 \leftarrow Q^0 \cup \{i\}$
15: **else** $|S_i^1| \geq \frac{1}{2}\left(1 - \frac{1}{k}\right)|S_i|$ **then** $S_{i+1} \leftarrow S_i^1$, $Q^1 \leftarrow Q^1 \cup \{i\}$
16: **if** $|Q^0| \geq k$ or $|Q^1| \geq k^3$ **then**
17: **break**
18: $i \leftarrow i + 1$
19: **return** Q^0 and Q^1

Since $p < k$, we have $|S_0| \geq \frac{n}{k}$, giving $|S_{k,k^3}| \geq \left(\frac{1}{2k}\right)^k \left(\frac{1}{2}\left(1 - \frac{1}{k}\right)\right)^{k^3} \frac{n}{k}$. Using the standard inequality $(1 - \frac{1}{k})^{k^3} \geq e^{-k^2}$ for large k, and the condition $|S_{k,k^3}| \leq 4$, we have $n \leq k^{k+1} \cdot 2^{k^3+k+2} \cdot e^{k^2}$. This contradicts our assumption that $n > k^{k+1} \cdot 2^{k^3+k+2} \cdot e^{k^2}$. Therefore, either $|Q^0| \geq k$ or $|Q^1| \geq k^3$ must hold. ◁

We now compute a pair of sets U^0 and U^1 as follows: $U^0 = \{(x_{2i}, x_{2i+1}) \mid i \in Q^0\}$ and $U^1 = \{(x_{2i}, x_{2i+1}) \mid i \in Q^1\}$, where Q^0 and Q^1 be the set returned by Algorithm 1.

▷ Claim 11. If $|Q^0| \geq k$ then $\psi(G) \geq k$.

Proof. Since $|Q^0| \geq k$, there are at least k tuples of the form $(x_{2i}, x_{2i+1}, d_{x_{2i},x_{2i+1}})$ for $i \in [|Q^0|]$. For each such tuple, let x_h denote the vertex between x_{2i} and x_{2i+1} that is not adjacent to $d_{x_{2i},x_{2i+1}}$. We assign a new color c_i to the pair $(x_h, d_{x_{2i},x_{2i+1}})$, thereby assigning at least k new colors. We now show that this coloring forms a partial $|Q^0|$-complete coloring. From the construction of Q^0 in Step 14 of the algorithm, we know that each vertex $d_{x_{2i},x_{2i+1}}$, for $i \in [|Q^0|]$, is adjacent to both x_{2j} and x_{2j+1} for all $j > i$. Thus, for any pair of colors c_i and c_j with $i > j$, the vertex $d_{x_{2j},x_{2j+1}}$ (colored with c_j) is adjacent to both x_{2i} and x_{2i+1}, at least one of which is colored with c_i. Therefore, there is an edge between every pair of color classes assigned in Step 7. By applying Lemma 6, this partial $|Q^0|$-complete coloring can be extended to a complete coloring of size at least $|Q^0|$. ◁

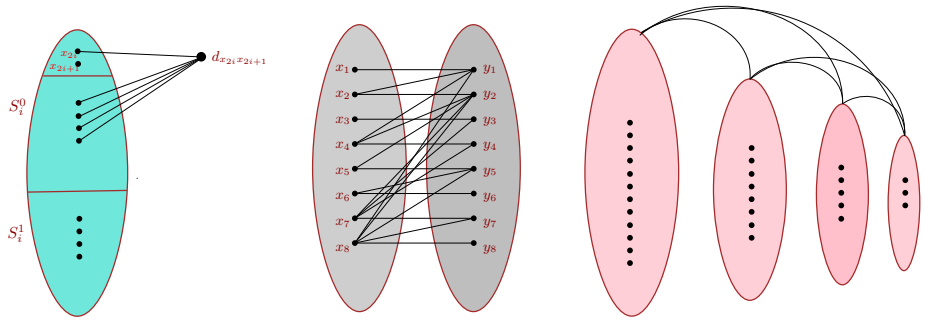


Figure 1 Figure depicting a partition of S_i in Algorithm 1, an instance of a semi-independent matching and a greedy independent partition respectively.

▷ Claim 12. If $|Q^1| \geq k^3$ then $\psi(G) \geq k$.

Proof. Let c' be the color that appears most frequently among the colors assigned to the vertices $d_{x_{2i},x_{2i+1}}$ for all $(x_{2i}, x_{2i+1}) \in U^1$. Define the subset $U' \subseteq U^1$ as the set of all pairs $(x_{2i}, x_{2i+1}) \in U^1$ such that $c(d_{x_{2i},x_{2i+1}}) = c'$. Since the graph G is colored using at most k colors, it follows by the pigeonhole principle that $|U'| \geq k^2$. We now construct a semi-independent matching from U'. For each pair $(x_{2i}, x_{2i+1}) \in U'$, include the vertex $d_{x_{2i},x_{2i+1}}$ and the one among x_{2i} and x_{2i+1} to which it is adjacent. Note that, by the definition of Q^1, any vertex x_j in U^1 with $j > 2i+1$ is not adjacent to $d_{x_{2i},x_{2i+1}}$. Therefore, the matching constructed in this way is semi-independent and has size at least $k^2 \geq \binom{k}{2}$. Hence, if $|Q^1| \geq k^3$, then we can compute a semi-independent matching of size at least $\binom{k}{2}$. By applying Lemma 8, we conclude that $\psi(G) \geq k$. ◁

The correctness of Step 3 follows from Lemma 17. In Claim 10, we show that if the Algorithm 1 outputs the pair of sets Q_0 and Q_1 then either $|Q^0| \geq k$ or $|Q^1| \geq k^3$. In both the cases, we have that $\psi(G) \geq k$ (by Claim 11 and 12). Hence we are done with showing that if the given graph G is irreducible and more than $k^{k+1}2^{k^3+k+2}e^{k^2}$ vertices then $\psi(G) \geq k$. This completes the proof. ◀

Recall that R was an equivalence relation defined as follows: for a vertex $v \in V(G)$, the class R_v contains the vertices $\{u \mid N_G(u) = N_G(v)\}$.

▶ **Lemma 13** ([10, Theorem 3.3]). *Given a graph G and an integer k, in time $\mathcal{O}(|E(G)|)$, we can*

- *determine if G has at least $f(k)$ equivalence classes, or*
- *build all the equivalence classes.*

Now we define a rule that bounds the number of vertices to each equivalence class.

▶ **Reduction Rule 2.** *If $|R_v| \geq k+1$, then delete v. Return the instance $(G - v, k)$.*

The correctness of the Reduction Rule 2 follows from the following claim.

▷ Claim 14. Reduction Rule 2 is safe.

Proof. In the forward direction, assume that (G, k) is a YES-instance of Achromatic Number. Then, $\psi(G) \geq k$. If $\psi(G) \geq k+1$ then $\psi(G - v) \geq k$ trivially holds, as deleting any vertex can decrease the achromatic number by at most one.

So we can assume that $\psi(G) = k$. Let R_v be an equivalence class with at least $k+1$ vertices $u_1, u_2, \ldots, u_{k+1}$. Without loss of generality, assume that $v = u_{k+1}$. By the definition of an equivalence class, we have $\forall i, j \in [k+1]$, $N_G(u_i) = N_G(u_j)$. On the contrary, assume

that $\psi(G-v) < k$. Let $C_1, \ldots, C_k$ be the color classes defined by the complete coloring of G. More precisely, for each $i \in [k]$, the set C_i is the set of all vertices in G with color i. Additionally, assume that $v \in C_k$. Now, if for all $i \in [k-1]$, we have an edge between a pair of vertices in C_i and C_k, excluding v, then we are done. Else, there exists a color class C_i such that, on removal of $v = u_{k+1}$, there is no edge between any pair of vertices in C_i and C_k. Assume that u is a vertex in C_i such that $(u, v) \in E(G)$. Note that in this case the k vertices $u_1, \ldots u_k$ are present in the remaining color classes $C_1, \ldots C_{k-1}$.

Thus by the pigeon hole principle, there exist u_ℓ and $u_{\ell'}$ in R_v, with $\ell \neq \ell' \in [k]$, which belong to the same color class, say $C_j, j \in [k-1]$. Because $N_G(u_\ell) = N_G(u_{\ell'})$, removing u_ℓ from C_j will not affect the achromatic number of G as $u_{\ell'}$ serves the same purpose as u_ℓ. Now, we can add u_ℓ to C_t. Since u_ℓ and $v = u_{k+1}$ belong to the same equivalence class, they are not connected by an edge and therefore u_ℓ, along with the vertices of C_t forms an independent set. The edge (u, u_ℓ) ensures an edge between the color classes C_i and C_t, and hence the achromatic number of the graph $G - v$ is k. Thus, (G', k) is also a YES-instance.

In the backward direction, let (G', k) be a YES-instance of Achromatic Number. Then, $\psi(G') \geq k$. Let $C_1, \ldots, C_{k'}$, where $k' \geq k$ be the partition of color classes corresponding to a complete coloring of G'. If for some $i \in [k']$, v does not have a neighbor in the vertices corresponding to C_i, then we can assign the color i to v and get a complete coloring of G with at least k colors. Otherwise, for each $i \in [k']$, the color class C_i contains a vertex from $N_G(v)$. Then, we assign a new color $k'+1$ to v and obtain a complete coloring of G with $k'+1$ colors. Hence, in both cases, we obtain a complete coloring on G with at least k colors. Thus, (G, k) is a YES-instance. ◁

▶ **Lemma 15.** *If the number of equivalence classes in G is strictly greater than $k^{k+1}2^{k^3+k+2}e^{k^2}$, then $\psi(G) \geq k$.*

Proof. Let $V(G)$ be partitioned into q equivalence classes. Assume $q > k^{k+1}2^{k^3+k+2}e^{k^2}$. Construct a subgraph $G[S]$, where $S \subseteq V(G)$ contains exactly one arbitrarily chosen vertex from each equivalence class. Clearly, $|S| = q$. Now we show that no two distinct vertices in S have the same open neighborhood. Suppose, for contradiction, that there exist distinct vertices $u, v \in S$ such that $N(u) = N(v)$. Then, by the definition of the equivalence relation, u and v must belong to the same equivalence class. However, since S contains exactly one vertex from each equivalence class, this contradicts the assumption that both u and v are in S. Therefore, all vertices in S have distinct open neighborhoods. Therefore, $G[S]$ is an irreducible graph. Since $q > k^{k+1}2^{k^3+k+2}e^{k^2}$, it follows that $|S| > k^{k+1}2^{k^3+k+2}e^{k^2}$. By applying Lemma 9, we obtain $\psi(G[S]) \geq k$. This gives a partial complete coloring of G with at least k colors. Finally, by applying Lemma 6, we conclude that $\psi(G) \geq k$. ◀

Let (G, k) be an instance where none of Reduction Rules 1 and 2 is applicable. Then each equivalence class is bounded by k. This imply the following lemma.

▶ **Lemma 16.** *Let (G, k) be an instance where none of Reduction Rules 1 and 2 is applicable. If the number of equivalence classes in G is at most $k^{k+1}2^{k^3+k+2}e^{k^2}$, then $|V(G)| \leq k^{k+2}2^{k^3+k+2}e^{k^2}$.*

▶ **Lemma 17.** *For any graph G, if size of greedy independent partition is at least k, then $\psi(G) \geq k$.*

Proof. We construct a *greedy independent partition* of G, say $I_1, I_2, \ldots, I_p$. We color each independent set with a different color. Since the size of the *greedy independent partition* is at least k, we use at least k colors. Now, from the construction of the *greedy independent*

partition, observe that all vertices in G outside I_1 must be adjacent to at least one vertex in I_1. Similarly, for each $i > 1$, the neighborhood of all vertices in I_i includes all vertices in $\bigcup_{j=i+1}^{p} I_j$. Thus, there exists no pair of independent sets in a *greedy independent partition* of G such that their union is also an independent set. Hence, our coloring is a complete coloring of size at least k, that is, $\psi(G) \geq k$. ◀

FPT algorithm for Achromatic Number**.** We first compute a kernel of size at most $k^{k+2}2^{k^3+k+2}e^{k^2}$ in $\mathcal{O}(m)$ time (by Theorem 5). It is easy to observe that if a graph admits a complete coloring with at least k colors, then there exists an induced subgraph $H \subseteq G$ with at most $\binom{k}{2}$ vertices that also admits a complete k-coloring. Therefore, on the kernelized instance, we can perform a brute-force search: enumerate all subsets of at most $\binom{k}{2}$ vertices and check whether any of them admits a complete coloring with at least k colors. Each such check can be performed in $\mathcal{O}(k^2)$ time. If such a subgraph is found, we return `YES` for the original instance (by Lemma 6); otherwise, we return `NO`. The kernelization step takes $\mathcal{O}(m)$ time, and the brute-force step takes at most $\binom{k^{k+2}2^{k^3+k+2}e^{k^2}}{\binom{k}{2}} \cdot \mathcal{O}(k^2) = 2^{\mathcal{O}(k^5)}$ time. Hence, the total running time of the algorithm for solving Achromatic Number is $2^{\mathcal{O}(k^5)} + \mathcal{O}(m)$.

▶ **Theorem 1.** Achromatic Number *can be solved in* $2^{\mathcal{O}(k^5)} + \mathcal{O}(m)$ *time.*

4 Parameterized by Vertex Cover

In this section, we study the parameterized complexity of Achromatic Number with respect to a structural parameter that is a modulator to edgeless graphs (also known as vertex cover). We call this version of the problem Achromatic Number/VC which is formally defined below.

Achromatic Number/VC **Parameter:** ℓ
Input: An undirected graph G, a set $S \subseteq V(G)$ of size ℓ such that $G - S$ is an independent set and a positive integer k.
Question: Is $\psi(G) \geq k$?

The following is the main result of this section.

▶ **Theorem 4.** Achromatic Number/VC *can be solved in* $2^{\mathcal{O}(\ell^2)} \cdot n^{\mathcal{O}(1)}$ *time.*

▶ **Observation 18.** *If G has a vertex cover of size at most ℓ, then $\psi(G) \leq (\ell + 1)$.*

Proof. Let S be a vertex cover of size at most ℓ in G. Suppose, for a contradiction, $\psi(G) > (\ell + 1)$. Then, there exists a complete coloring $\mathcal{C}$ using at least $\ell + 2$ colors. Since $|S| \leq \ell$, at most ℓ color classes of $\mathcal{C}$ contain the vertices of the set S. This implies that there are at least two color classes, say C_i and C_j, which do not contain any vertices of S. Now the subgraph of G induced by the vertices of $C_i \cup C_j$ is an independent set, which implies that there is no edge across the color classes C_i and C_j. This contradicts the fact that $\mathcal{C}$ is a complete coloring. ◀

An input instance of our problem is (G, S, k, ℓ), where S is a vertex cover of size at most ℓ. Furthermore, assume that (G, S, k, ℓ) is a `YES` instance and $\Pi = (X_1, X_2, \ldots, X_k)$ is a hypothetical solution where Π is a partition of vertices into k independent sets $X_1, X_2, \ldots, X_k$ and, for each pair i, j there is a pair of vertices $u \in X_i$ and $v \in X_j$ such that $(u, v) \in E(G)$.

Our ultimate objective is to get Π. To do this, we try to obtain as much information as possible about Π in time $\ell^{\ell} \cdot n^{\mathcal{O}(1)}$. Observe that if S is an empty set, then (G, S, k, ℓ) is a YES instance if and only if $k = 1$, and we can obtain Π by simply setting $X_1 = V(G)$. In what follows, we assume that we are given an input (G, S, k, ℓ) and a partition $\pi = (Y_1, Y_2, \ldots, Y_k)$ of S, and our problem is to test whether π can be *extended* to the desired partition Π. More specifically, we test whether there is a feasible solution, that is, partition $\Pi = (X_1, X_2, \ldots, X_k)$ of $V(G)$ such that $Y_i \subseteq X_i$, for each $1 \leq i \leq k$. This leads us to the following problem.

DISJOINT ACHROMATIC NUMBER/VC **Parameter:** ℓ
Input: A graph G, a set $S \subseteq V(G)$ of size at most ℓ such that $G - S$ is an independent set, an integer k, and a partition $\pi = (Y_1, Y_2, \ldots, Y_k)$ of S.
Question: Does there exist a solution $\Pi = (X_1, X_2, \ldots, X_k)$ with the requisite properties that extends π?

We use (G, S, k, ℓ, π) to denote an instance of DISJOINT ACHROMATIC NUMBER/VC.

Our next lemma formally proves our discussion by showing that ACHROMATIC NUMBER/VC and DISJOINT ACHROMATIC NUMBER/VC are FPT equivalent. That is, ACHROMATIC NUMBER/VC is FPT if and only if DISJOINT ACHROMATIC NUMBER/VC is FPT.

▶ **Lemma 19.** *Let G be a graph and S be a vertex cover of size at most ℓ. For any integer k, (G, S, k, ℓ) is a* YES*-instance of* ACHROMATIC NUMBER/VC *if and only if either (G, S, k, ℓ, π) is a* YES*-instance or there exists a non-empty set $X \subseteq V(G) \setminus S$ such that $|X| \leq \ell$ and $(G, S \cup X, k, 2\ell, \pi)$ is a* YES*-instance of* DISJOINT ACHROMATIC NUMBER/VC*, for some partition π of S or $S \cup X$ respectively.*

Proof. The backward direction is obvious due to the definition of DISJOINT ACHROMATIC NUMBER/VC. If either (G, S, k, ℓ, π) or $(G, S \cup X, k, 2\ell, \pi)$ for some $X \subseteq V(G) \setminus S$ is YES, then by definition we have a feasible solution with at least k colors that extends π.

In the forward direction, let (G, S, k, ℓ) be a YES instance of ACHROMATIC NUMBER/VC, then there exists a partition of G, say $C_1, \ldots, C_t$, with $t \geq k$, such that between any two pairs of color classes C_i and C_j, $i \neq j \in [t]$, there exist vertices $u \in C_i$ and $v \in C_j$ with $(u, v) \in E(G)$. Observe that there can be at most one color class that does not contain any vertices from S. Now there are two cases.

Case 1. Every color class C_i intersects S. In this case, we define a partition $\pi = (Y_1, Y_2, \ldots, Y_t)$ of S as follows. For each $i \in [t]$ the set $Y_i := S \cap C_i$. Hence (G, S, k, ℓ, π) is a YES instance.

Case 2. There is a color class that does not intersect S. Wlog, assume that C_1 is such a color class. Note $t \leq \ell + 1$. Observe that for every set C_i, $2 \leq i \leq t$, there is a vertex in C_1 that has a neighbor in C_i. There may be many vertices that has a neighbor in C_i but we choose an arbitrary vertex $u_i \in C_1$ that has a neighbor in C_i. In this process, for each i, $2 \leq i < [t]$, we find a vertex $u_i \in C_1$. Let $X = \{u_2, \ldots, u_q\}$, where $q \leq t$. As $q \leq t \leq \ell + 1$, so $|X| \leq \ell$. Clearly, $S \cup X$ is a vertex cover of size at most 2ℓ. Now we define a partition $\pi = (Y_1, Y_2, \ldots, Y_k)$ of $S \cup X$ as follows. For $i = 1$, define $Y_1 := X$, for all other i, $2 \leq i \leq [t]$ we define the set $Y_i := S \cap C_i$. Hence $(G, S \cup X, k, 2\ell, \pi)$ is a YES instance. ◀

Next, we aim to generate a collection $\mathcal{I}$ of $f(\ell)$ many instances for DISJOINT ACHROMATIC NUMBER/VC from an input instance (G, S, k, ℓ) of ACHROMATIC NUMBER/VC. First, for each partition π of S, we include an instance (G, S, k, ℓ, π) into $\mathcal{I}$. Next, consider the equivalence relation R defined for Lemma 13. Given that $|S| \leq \ell$, it follows that the number

of equivalence classes is at most 2^ℓ. Thus, for any pair of vertices u, v in the same equivalence class, we have $N(u) = N(v)$. Let Z^* be a set of vertices formed by arbitrarily choosing a vertex from each equivalence class. Since the number of equivalence classes is at most 2^ℓ, it follows that $|Z^*| \leq 2^\ell$. Now for each subset $Z \subseteq Z^*$ where $|Z| \leq \ell$, and for each partition π of $S \cup Z$, we add an instance $(G, S \cup Z, k, 2\ell, \pi)$ to $\mathcal{I}$. Therefore, the total number of instances is bounded by $\ell^\ell + \binom{2^\ell}{\ell} \cdot (\ell+1)^{2\ell} \leq 2^{\mathcal{O}(\ell^2)}$. The following claim is derived from the arguments of the proof of Lemma 19 which basically tells that these two problems are FPT equivalent.

▷ Claim 20. (G, S, k, ℓ) is YES instance if and only if one of the instances of $\mathcal{I}$ is YES.

4.1 Algorithm for Disjoint Achromatic Number/vc

Let (G, S, k, ℓ, π) be an instance of Disjoint Achromatic Number/vc, where $\pi = (Y_1, Y_2, \ldots, Y_k)$. Our algorithm works as follows. First, the algorithm performs a simple sanity check reduction rule. In essence, it checks whether π is valid.

▶ **Reduction Rule 3.** *Return that (G, S, k, ℓ, π) is a* NO *instance of* Disjoint Achromatic Number/vc*, if one of the following holds:*

1. *$S = \emptyset$ and $k \geq 2$.*
2. *There is a set Y_i in $\pi = (Y_1, \ldots, Y_k)$ such that Y_i is not an independent set.*
3. *There is a pair of sets Y_i and Y_j in $\pi = (Y_1, \ldots, Y_k)$ such that $Y_i \cup Y_j \cup (V - S)$ is an independent set.*

▶ **Lemma 21.** *Reduction Rule 3 is safe.*

Proof.

1. If $S = \emptyset$, then G is an independent set. If there are at least 2 color classes, then there is no edge going across any pair of color classes. Therefore, (G, S, k, ℓ, π) is a NO instance of Disjoint Achromatic Number/vc for $k \geq 2$.
2. If there is a set Y_i in $\pi = (Y_1, \ldots, Y_k)$ such that Y_i is not an independent set, then if we extend π to Π, a partition of G, we get $X_i \supseteq Y_i$ which is not an independent set. This violates the property of complete coloring.
3. If there are two sets Y_i and Y_j in π, then an edge that goes across the pair of color classes Y_i and Y_j must have endpoints in Y_i and Y_j, Y_i and $V - S$ or Y_j and $V - S$. However, the graph induced on $Y_i \cup Y_j \cup (V - S)$ is an independent set. Therefore, (G, S, k, ℓ, π) is a NO instance of Disjoint Achromatic Number/vc. ◀

Now we describe our algorithm. We first find the bad pairs in $[k] \times [k]$. We say that a pair (i, j) is a bad pair if and only if $Y_i \cup Y_j$ induces an independent set in G. It is easy to observe that the number of bad pairs is at most k^2.

We now construct two graphs; one is the *host graph*, denoted by G_1, and another is the *pattern graph* G_2 satisfying the condition as follows: G_1 has a subgraph isomorphic to G_2 if and only if (G, S, k, ℓ, π) is an YES-instance. Let b be the number of bad pairs in $[k] \times [k]$ and $t = |V \setminus S|$. See Figure 2 and Figure 3 for an illustration of the construction.

Construction of host graph G_1

1. We add two sets $A_1 = \{w_1, \ldots, w_b\}$ and $B_1 = \{u_1, \ldots, u_b\}$, each of the b vertices in $V(G_1)$ corresponding to each bad pair in $[k] \times [k]$.
2. We add a set $I_1 = \{v_1, \ldots, v_t\}$ of t vertices to $V(G_1)$, one for each vertex in $V \setminus S$.

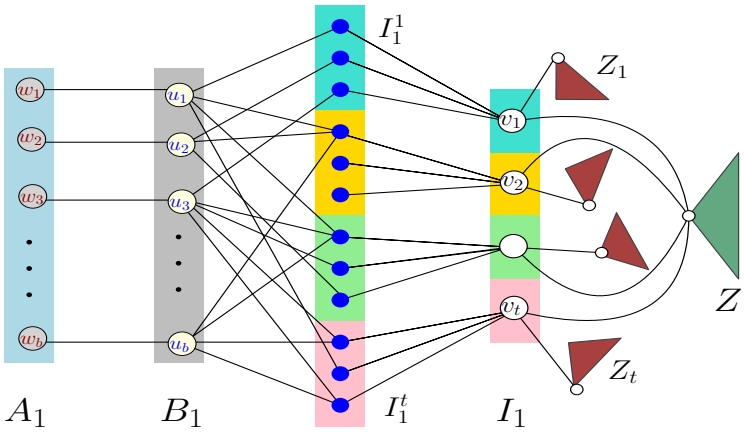


Figure 2 An illustration of the construction of host graph.

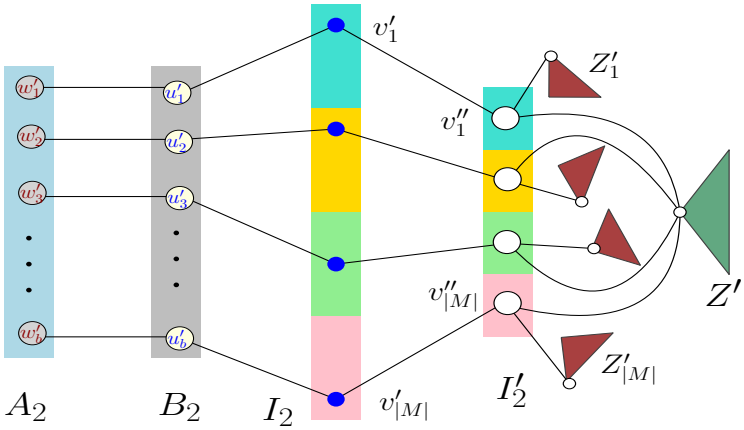


Figure 3 An illustration of the construction of pattern graph.

3. We add a set $I_1^i = \{v_i^1, \ldots, v_i^k\}$ of k vertices for each $1 \leq i \leq t$ in $V(G_1)$.
4. We add t cliques $Z_1, \ldots, Z_t$, each of size 50 in G_1. For each clique Z_i take an arbitrary vertex in it and make it adjacent to $v_i \in I_1$.
5. We add a clique Z of size 100 to G_1. Choose an arbitrary vertex in Z and make it adjacent to each vertex in I_1.
6. Add the set $\{(u_i, w_i) \colon i \in [b]\}$ of edges to $E(G_1)$.
7. Add the set $\{(v_i, v_i^j) \colon i \in [t], j \in [k]\}$ of edges to $E(G_1)$.
8. For any pair of integers i and j, we add an edge between $v_i^j \in I_1^i$ and a vertex $u \in B_1$, where u is a bad pair corresponding to (c, j) or (j, c) if (i) v_i has no neighbor in Y_j and (ii) v_i has a neighbor in Y_c.

Observe here that in step 8 when we add an edge between $v_i^j \in I_1^i$ and a vertex $u \in B_1$, if the vertex v_i is assigned a color j, that is, v_i is added to the partition Y_j then it satisfies the bad pair corresponding to u. Here, we say that a vertex satisfies a bad pair when its addition to one of the partitions of the pair introduces an edge between the pair. Our goal is to check if there exists an assignment of colors to the vertices in $G \setminus S$ that satisfies all the bad pairs.

We create 2^b many pattern graphs as follows: Let M be a minimal set of vertices in $\bigcup_{i=1}^{t} I_1^i$ such that (i) $M \cap I_1^i \leq 1$ for each $1 \leq i \leq t$, (ii) $N(u) \cap M \neq \emptyset$ for any $u \in B_1$. Clearly $|M| \leq b$. We guess the value of $|M|$. Then we guess $|M|$ numbers such that their sum is b. We use the following fact to bound the number of such partitions.

▶ **Fact 1.** *For any positive integers n and k the number of k tuples of positive integers whose sum is n equals $\binom{n-1}{k-1} \leq 2^n$.*

Let us guess the value of $|M|$ and $|M|$ numbers $n_1, n_2, \ldots, n_{|M|}$ whose sum equals b. Each value n_i is essentially the number of bad pairs that a vertex will satisfy privately. We will later describe the notion of satisfying privately. In the following, we describe how to construct a pattern graph for such a guess. Putting $n = b$ in Fact 1, we can decide that the number of such partitions is at most 2^b. Let $\mathcal{H}$ denote the set of all pattern graphs.

Construction of pattern graph $G_2 \in \mathcal{H}$: For $n_1 + n_2 + \ldots + n_{|M|} = b$

1. We add two sets $A_2 = \{w'_1, \ldots, w'_b\}$ and $B_2 = \{u'_1, \ldots, u'_b\}$, each of the b vertices in $V(G_2)$.
2. We add a set $I_2 = \{v'_1, v'_2, \ldots, v'_{|M|}\}$ of $|M|$ vertices to $V(G_2)$.
3. We add a set $I'_2 = \{v''_1, v''_2, \ldots, v''_{|M|}\}$ of $|M|$ vertices to $V(G_2)$.
4. We add $|M|$ cliques $Z'_1, \ldots, Z'_{|M|}$, each of size 50 in G_2. For each clique Z'_i take an arbitrary vertex in it and make it adjacent to $v''_i \in I'_2$.
5. We add a clique Z' of size 100 in G_2. Choose an arbitrary vertex in Z' and make it adjacent to each vertex in I'_2.
6. Add the set $\{(u'_i, w'_i) \colon i \in [b]\} \bigcup \{(v'_i, v''_i) \colon i \in [|M|]\}$ of edges to $E(G_2)$.
7. Add the set $\{(v'_i, u'_j) \colon i \in [|M|], (n_1 + \ldots + n_{i-1}) + 1 \leq j \leq (n_1 + \ldots + n_i), n_0 = 0\}$ of edges to $E(G_2)$.

▷ Claim 22. $\mathsf{tw}(G_2) = \mathcal{O}(1)$.

Proof. Consider the graph G'_2 without copies of cliques K_{100} and K_{50}. Let (T, β) be the tree decomposition of G'_2. The treewidth of G'_2 is 1 as it is a tree. We add the cliques K_{100} and K_{50} to every bag of T. This gives us a tree decomposition of G_2 with a constant treewidth. ◁

▶ **Observation 23.** $|V(G_2)| = \mathcal{O}(k^2)$.

Proof. We know that $|A_2| = |B_2| \leq b \leq k^2$, and $|I_2| = |I'_2| \leq b \leq k^2$. There are $|M|$ cliques of size 50 each and a clique of size 100. Thus, $|V(G_2)| \leq 4k^2 + 50k^2 + 100 = \mathcal{O}(k^2)$. ◀

The following lemma proves the correctness of our algorithm.

▶ **Lemma 24.** (G, S, k, ℓ, π) *is a* YES *instance of* Disjoint Achromatic Number/VC *if and only if there exists a subgraph of G_1 which is isomorphic to G_2 for some $G_2 \in \mathcal{H}$.*

Proof. Assume (G, S, k, ℓ, π) is a YES instance of Disjoint Achromatic Number/VC. Then there is a partition π of S that can be extended to a partition Π of G (each part of a partition corresponds to a color class) such that there is an edge across any pair of color classes in the partition Π. We show the existence of a pattern graph G_2 that is isomorphic to a subgraph of G_1. We construct a subgraph G'_1 of graph G_1 as follows:

- In the graph G_1, for each set I_1^i, keep only the vertex v_i^J where the vertex corresponding to v_i in G gets color $J \in [k]$ in the solution Π and delete other vertices of I_1^i
- Delete the edges of v_i^J to those vertices of B_1 that are adjacent to some vertex v_p^Q , where $p \leq i - 1$ and $Q \in [k]$. Now no two vertices v_i^J, v_p^Q have any common neighbor.
- Let n_i be the number of neighbors of v_i^J in B_1, for $i \in [t], J \in [k]$.
- If $n_i = 0$, then we also delete v_i^J

- After this step let $\{v_1^{J_1}, v_2^{J_2}, \dots v_h^{J_h}\}$ be the vertices remaining from vertices of the form v_i^J
- Now in A_1, B_1 keep only the vertices w_i, u_i such that u_i has neighbor of the form v_p^J.
- In the set I_1 keep only vertices v_p such that it has a neighbor of the form v_p^J. Let the remaining vertices be $\{v_1, \dots v_h\}$.
- Delete all the isolated K_{50}.

We will show that this constructed subgraph G_1' is isomorphic to a pattern graph $G_2 \in \mathcal{H}$ corresponding to the values $n_1 + n_2 + \ldots + n_h = b$. We show the isomorphism $f : V(G_1') \to V(G_2)$ as follows.

- For every $v_i^{J_i} \in G_1'$, we map $f(v_i^{J_i}) = v_i'$.
- Now both $v_i^{J_i}, v_i'$ have n_i neighbors in B_1, B_2 respectively. Suppose $v_i^{J_i}$ is adjacent to $\{u_{i_1}, u_{i_2}, \dots u_{i_{n_i}}\} \subset B_1$. Then we have $f(u_{i_j}) = u'_{(1+\dots n_{i-1})+j}$
- Correspondingly $f(w_{i_j}) = w'_{(1+\dots n_{i-1})+j}$
- For $v_i \in I_1$ we have $f(v_i) = v_i''$.
- The K_{50} which is neighbor of $v_i \in I_1$ gets mapped to the K_{50} which is neighbor of v_i''
- K_{100} in G_1' gets mapped to the K_{100} in G_2

Conversely, suppose that there is a subgraph of G_1 that is isomorphic to some pattern graph $G_2 \in \mathcal{H}$. Since K_{100} is the largest clique in G_1, it is isomorphic to the K_{100} in G_2. The set I_2' in G_2 is isomorphic to a subset of the set I_1 in G_1, as the vertices of both sets are neighbors of the cliques K_{100} and copies of K_{50}. Every vertex $v_i'' \in I_2'$ is adjacent one vertex in I_2. Thus at most one vertex from each I_1^i is mapped to a vertex in I_2. Now for each set I_1^i consider the vertex that was mapped to a vertex in I_2. It has neighbors in B_1 and hence this subset of B_1 is mapped to B_2 in G_2. Similarly, the corresponding subset of A_1 is mapped to A_2 in G_2. Since, the pattern graph G_2 is isomorphic to a subgraph of G_1 and there exists at most one vertex v_i^J in each I_1^i in G_1 that has n_i private neighbors in B_1. Since $n_1 + \dots n_{|M|} = b$, the selected vertices in I_1^i in total have b neighbors in B_1, which implies that each vertex in B_1 has a unique neighbor in I_1^i. Thus, the neighborhood of the mapped vertices in I_1^i in B_1 is exactly B_1. An edge (u_r, v_i^J) in G_1 implies that the vertex v_i when colored with color J satisfies the bad pair corresponding to r in B_1. Thus, all the bad pairs in B_1 are being satisfied. Therefore, we can conclude that (G, S, k, ℓ, π) is a `YES` instance.

◀

▶ **Fact 2.** [1, Theorem 6.3] *Let H be a directed or an undirected graph on k vertices with treewidth tw. Let $G = (V, E)$ be a (directed or undirected) graph. A subgraph of G isomorphic to H, if one exists, can be found in the expected time $2^{\mathcal{O}(k)} \cdot |V|^{tw+1}$ and in the worst-case time $2^{\mathcal{O}(k)} \cdot |V|^{tw+1} \cdot \log |V|$.*

We describe our algorithm in Figure 4. Note that by the construction of our pattern graph and the proof of Lemma 24 our algorithm will also be able to return a satisfying coloring in the case of a `YES` instance.

Running time. First, we analyze the running time of the algorithm for Disjoint Achromatic Number/vc. The number of guesses in step 2 is at most $b2^b \leq k^2 2^{k^2}$. The running time of the algorithm at step 4 is $2^{\mathcal{O}(k^2)} n^{\mathcal{O}(1)}$ from Fact 2. Thus, the overall running time of the algorithm for Disjoint Achromatic Number/vc is $k^2 2^{k^2} 2^{\mathcal{O}(k^2)} n^{\mathcal{O}(1)} = 2^{\mathcal{O}(k^2)} n^{\mathcal{O}(1)} = 2^{\mathcal{O}(\ell^2)} n^{\mathcal{O}(1)}$ since $\ell \leq k+1$ from Observation 18. Note that the proof works even if the cliques in the host and pattern graph are K_3 and K_5 in place of K_{50} and K_{100}, respectively. We assumed a large clique for clarity of understanding. Now, in the algorithm for Achromatic Number/VC, we make ℓ^ℓ many guesses of the partitions of S and run the algorithm for

DISJOINT ACHROMATIC NUMBER/VC ($I = (G, S, k, \pi)$)
(It checks if (G, S, k, π) is a YES instance)

1. Construct the host graph G_1.
2. Guess a value of $|M|$, ($|M| \leq b$), and a partition of b into $|M|$ positive integers $\{n_1, \ldots, n_{|M|}\}$.
3. For each such guess construct a pattern graph G_2.
4. Use the algorithm from Fact 2 to check if a subgraph of G_1 is isomorphic to G_2 .
5. If a subgraph of G_1 is isomorphic to G_2 output YES instance .
6. Return NO if none of the pattern graphs G_2 is isomorphic to a subgraph of G_1

Figure 4 Algorithm for DISJOINT ACHROMATIC NUMBER/VC.

DISJOINT ACHROMATIC NUMBER/VC for each guess. Thus, the overall running time of the algorithm for DISJOINT ACHROMATIC NUMBER/VC is $\ell^{\ell} \cdot 2^{\mathcal{O}(\ell^2)} \cdot n^{\mathcal{O}(1)} = 2^{\mathcal{O}(\ell^2)} \cdot n^{\mathcal{O}(1)}$. Thus, we get the following theorem.

▶ **Theorem 4.** ACHROMATIC NUMBER/VC *can be solved in* $2^{\mathcal{O}(\ell^2)} \cdot n^{\mathcal{O}(1)}$ *time.*

5 Kernel for d-degenerate graphs

In this section, we obtain a polynomial kernel for ACHROMATIC NUMBER when the graph is d-degenerate. First, we state a crucial theorem that we use to design the polynomial kernel for graphs that are d-degenerate. Then, we prove a few lemmatas to show the correctness of our kernelization algorithm. We start with the following lemma, which asserts that if a graph contains a *large* induced matching, then the achromatic number of the graph is also large.

▶ **Lemma 25.** *Let G be an undirected graph. For any positive integer k, if G has an induced matching of size at least k^2, then $\psi(G) \geq k$.*

Proof. Let G be an undirected graph and k be a positive integer. Let us assume that the graph G contains an induced matching M of size $\binom{k}{2}$. The set $V(M)$ denotes the set of vertices in G that are saturated by the matching M. First, we give a complete coloring using k colors on the matched vertices. Note that there are exactly $\binom{k}{2} \leq k^2$ distinct pairs of colors. We color the end points of the edges of M in the following greedy way. We say that a pair (i, j) is unassigned if there is no edge $e = (u, v)$ in M such that color $(u) = i$ and color $(v) = j$. Furthermore, an edge $e = (u, v)$ is uncolored if we have not assigned any color to u and v. Initially, all edges in M are uncolored, and all pairs (i, j), where $1 \leq i < j \leq [k]$, are unassigned. In each step, we take an uncolored edge $e = (u, v) \in M$, an unassigned pair (i, j), chosen arbitrarily, and add the color i and j to u and v, respectively. We stop when there is no such pair and an uncolored edge exists. As both the number of pairs and edges in M is exactly $\binom{k}{2}$, for any $i, j \in [k]$ with $i < j$, we have exactly one edge whose end points are colored with colors i and j.

In the above process, we color exactly $2|M|$, that is, $2\binom{k}{2}$ vertices. For the remaining $n - 2\binom{k}{2}$ vertices in the graph, we do the following. Let t denote the number of colors used so far in the process. Obviously, $t \geq k$. We take an uncolored vertex $v \in V(G) \setminus V(M)$. If there is an $i \in [t]$ such that v does not have neighbors in the vertices that have already been colored i, we assign the color i to the vertex v. If no such i exists, then we give a new color to the vertex v. It is easy to see that this procedure does not violate the complete coloring property. As we use at least k colors, we get a complete coloring of G with at least k colors. ◀

Kernelization Algorithm$(G = (V, E), k)$
Given a d-degenerate graph G, it returns a kernel of size $k^{\mathcal{O}(d)}$.

1. Apply Reduction Rule 2 exhaustively to get G' with n' vertices.
2. If $|n'| \geq \frac{4k(k+1)}{3}(k^3 + 8d - 1)^{8d}$, return a trivial YES instance
3. Else, return the reduced graph G'.

Figure 5 Kernelization Algorithm for Achromatic Number on d-degenerate graphs.

▶ **Definition 26** (Strong system of distinct representatives)**.** *A system of distinct representatives* for the sets $S_1, S_2, \ldots, S_k$ is a k-tuple $(x_1, x_2, \ldots, x_k)$ where the elements x_i are distinct and $x_i \in S_i$, for all $i \in [k]$. In addition to that, if we have $x_i \notin S_j$ for all $i \neq j$, then such a system is called *strong*.

▶ **Theorem 27** ([16, Theorem 8.12])**.** *For a pair of integers r and t, in any family of more than $\binom{r+t}{t}$ sets of cardinality at most r, at least $t+2$ of its members have a strong system of distinct representatives.*

The following property of a d-degenerate graph follows directly from the definition.

▶ **Proposition 28.** *The number of edges in a d-degenerate graph is bounded by dn.*

Next, we give a lower bound on the number of low-degree vertices in a d-degenerate graph.

▶ **Lemma 29.** *Let G be a d-degenerate graph and c be a positive integer with $c > 2$. Then, G contains strictly more than $(\frac{c-2}{c})n$ vertices with degree at most cd.*

Proof. Let G be a d-degenerate graph on n vertices. By Proposition 28, the number of edges is at most dn. Therefore, the sum of the degrees of the vertices in G is bounded by $2dn$. Assume that there are at most $(\frac{c-2}{c})n$ vertices of degree at most cd in G. Then we have a set $U \subseteq V(G)$ of size at least $n - (\frac{c-2}{c})n = \frac{2n}{c}$ vertices of degree strictly more than cd. Now, the sum of the degrees of the vertices in U is strictly more than $\frac{2n}{c} \cdot cd = 2dn$, a contradiction. Therefore, there are strictly more than $(\frac{c-2}{c})n$ vertices of degree at most cd in G. ◀

Let us define a *greedy independent partition* of G as follows. Construct a partition of $V(G)$ into independent sets by iteratively finding maximal independent sets. Then, in such a partition, among all sets, choose a maximal independent, say I_1, which has the highest cardinality (if there is more than one such set, then arbitrarily choose one) and arbitrarily order the remaining sets in the partition keeping I_1 as the first element. We denote this ordered family of sets as a *greedy independent partition*. Note that a *greedy independent partition* of G can be constructed in $\mathcal{O}(m)$ time.

Now, we describe our kernelization algorithm. A brief description of our algorithm is given in Figure 5.

▶ **Theorem 3.** Achromatic Number *admits a kernel of size $\mathcal{O}(k^{24d+2})$ on d-degenerate graphs.*

Proof. First, we construct a *greedy independent partition* of G, say $I_1, I_2, \ldots, I_p$. If $p \geq k$, then from Lemma 17, we get a complete coloring of G of size at least k and return a trivial YES-instance. Else, we do as follows.

We apply Reduction Rule 2 exhaustively. Let G' be the reduced graph with n' vertices. Note that after this step, at most $k+1$ vertices can have the same open neighborhood. Since our graph G is d-degenerate, the reduced graph G' is also d-degenerate. From Lemma 29, we know that the number of vertices with degree at most $8d$ is at least $\frac{3n'}{4}$. Let us call vertices with degree at most $8d$ as low-degree vertices. We denote these vertices by V_{low}. Let $\mathcal{A} = \{A_1, A_2, \dots A_l\}$ be a *greedy independent partition* of $G'[V_{\text{low}}]$. If $|A_1| < \frac{3n'}{4k}$, then we have $l \geq k$ (as $|V_{\text{low}}| \geq \frac{3n'}{4}$). So, applying Lemma 17, we have $\psi(G'[V_{\text{low}}]) \geq k$ and that immediately implies $\psi(G') \geq k$, and we return a trivial YES instance. Otherwise, we have $|A_1| \geq \frac{3n'}{4k}$.

Let $\{v_1, v_2, \dots, v_{|A_1|}\}$ be the vertices in A_1 and $\{S_1, S_2, \dots, S_{|A_1|}\}$ be their corresponding open neighborhoods. Due to the exhaustive application of Reduction Rule 2 at least $\frac{|A_1|}{k+1}$ of these open neighborhoods are distinct. Let $\frac{|A_1|}{k+1} = q$ and $\{S_1, S_2, \dots, S_q\}$ be the pairwise distinct sets. Let $S = \bigcup_{i=1}^{q} \{S_i\}$. Thus, S is a family of more than $\frac{3n'}{4k(k+1)}$ sets, each of cardinality at most $8d$. Let k' be the integer such that

$$\binom{8d+k'-1}{k'-1} \leq \frac{3n}{4k(k+1)} \leq \binom{8d+k'}{k'} \tag{2}$$

By applying Theorem 27 on the family of sets S, we know at least $k'-1+2 = k'+1$ of its members have a strong system of distinct representatives. We match each representative vertex with its corresponding vertex in A_1. Let B be the set of those representative vertices. Basically B represents those set of vertices in $G - A_1$ such that we have a set of vertices $A' \subseteq A_1$ satisfying (i) $|B| = |A'|$, (ii) for each vertex $b \in B$ there exists a unique vertex $v_a \in A'$ such that $b \in N(v_a)$ and $b \notin N(v)$ for all $v \in A' \setminus \{v_a\}$. We compute a *greedy independent partition*, say $\mathcal{B} = \{B_1, B_2, \dots, B_j\}$ in $G'[B]$. Now we have the following cases.

- If $|B_1| < \frac{|B|}{k}$, then we have $j \geq k$. From Lemma 17, we get $\psi(G'[B]) \geq k$ and that immediately implies $\psi(G') \geq k$, and we return a trivial YES instance.
- Otherwise, we have $|B_1| \geq \frac{|B|}{k}$. So, we have an induced matching of size at least $\frac{|B|}{k}$ in G'.
  - If $\frac{|B|}{k} \geq k^2$, then we obtain an induced matching of size at least k^2. Then applying Lemma 25, we have $\psi(G') \geq k$. Hence, we return a trivial YES instance.
  - Otherwise, we have $\frac{|B|}{k} < k^2$.

Now $|B| \geq k'+1$ and $\frac{|B|}{k} < k^2$ together imply

$$k' \leq k^3 - 1 \tag{3}$$

Using a binomial formula, we have the following.

$$\binom{8d+k'}{k'} = \binom{8d+k'}{8d} \leq (8d+k')^{8d} \tag{4}$$

From Equation (2) and Equation (4) we have

$$\frac{3n'}{4k(k+1)} \leq \binom{8d+k'}{k'} \leq (8d+k')^{8d}$$

$$\Rightarrow (8d+k') \geq \left(\frac{3n'}{4k(k+1)}\right)^{\frac{1}{8d}}$$

$$\Rightarrow k' \geq \left(\frac{3n'}{4k(k+1)}\right)^{\frac{1}{8d}} - 8d \tag{5}$$

From Equation (3) and Equation (5), we get

$$\begin{aligned}(\frac{3n'}{4k(k+1)})^{\frac{1}{8d}} - 8d \leq k' < k^3 - 1 \\ n' < \frac{4k(k+1)}{3}(k^3 + 8d - 1)^{8d} \\ n' = \mathcal{O}(k^{24d+2}) \end{aligned} \tag{6}$$

Hence Achromatic Number admits a kernel of size $\mathcal{O}(k^{24d+2})$ on d-degenerate graph. ◀

6 Conclusion

In this paper, we do a parameterized reunion with Achromatic Number, and design an FPT algorithm with explicit running time on general graphs. We also study the problem with respect to structural parameterizations. Our work leaves several intriguing open questions.

1. Does Achromatic Number admit a polynomial kernel on general graphs?
2. We showed that Achromatic Number/VC can be solved in $2^{\mathcal{O}(\ell^2)} \cdot n^{\mathcal{O}(1)}$. We can show the following lower bound on the running time of an algorithm for Achromatic Number/VC.

   ▶ **Theorem 30 (♠).** *Unless* ETH *fails,* Achromatic Number/VC *cannot be solved in time* $2^{o(\ell)} \cdot n^{\mathcal{O}(1)}$*, where* ℓ *is the size of the vertex cover.*

   A natural question is whether we could close the gap between the upper and the lower bounds on the running time of Achromatic Number/VC.

References

1 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM (JACM)*, 42(4):844–856, 1995. doi:10.1145/210332.210337.

2 Katerina Asdre, Kyriaki Ioannidou, and Stavros D. Nikolopoulos. The harmonious coloring problem is np-complete for interval and permutation graphs. *Discret. Appl. Math.*, 155(17):2377–2382, 2007. doi:10.1016/J.DAM.2007.07.005.

3 V. Bhave. On the pseudoachromatic number of a graph. *Fundamenta Mathematicae*, 102(3):159–164, 1979.

4 Hans L. Bodlaender. Achromatic number is np-complete for cographs and interval graphs. *Inf. Process. Lett.*, 31(3):135–138, 1989. doi:10.1016/0020-0190(89)90221-4.

5 Béla Bollobás, Bruce A. Reed, and Andrew Thomason. An extremal function for the achromatic number. In Neil Robertson and Paul D. Seymour, editors, *Graph Structure Theory, Proceedings of a AMS-IMS-SIAM Joint Summer Research Conference on Graph Minors held June 22 to July 5, 1991, at the University of Washington, Seattle, USA*, volume 147 of *Contemporary Mathematics*, pages 161–165. American Mathematical Society, 1991.

6 Niall Cairnie and Keith Edwards. Some results on the achromatic number. *J. Graph Theory*, 26(3):129–136, 1997. doi:10.1002/(SICI)1097-0118(199711)26:3\%3C129::AID-JGT3\%3E3.0.CO;2-T.

7 Jianer Chen, Iyad A. Kanj, Jie Meng, Ge Xia, and Fenghui Zhang. On the pseudo-achromatic number problem. *Theor. Comput. Sci.*, 410(8-10):818–829, 2009. doi:10.1016/J.TCS.2008.11.010.

8 K. Edwards and C. McDiarmid. The complexity of harmonious colouring for trees. *Discrete Applied Mathematics*, 57(2-3):133–144, 1995. Copyright 2009 Elsevier B.V., All rights reserved. doi:10.1016/0166-218X(94)00100-R.

9 Keith J. Edwards. Achromatic number versus pseudoachromatic number: a counterexample to a conjecture of hedetniemi. *Discrete Mathematics*, 219(1):271–274, 2000. doi:10.1016/S0012-365X(00)00025-X.

10 Martin Farber, Gena Hahn, Pavol Hell, and Donald J. Miller. Concerning the achromatic number of graphs. *J. Comb. Theory, Ser. B*, 40(1):21–39, 1986. doi:10.1016/0095-8956(86)90062-6.

11 John P. Georges. On the harmonious coloring of collections of graphs. *Journal of Graph Theory*, 20(2):241–254, 1995. doi:10.1002/jgt.3190200213.

12 R. P. Gupta. The achromatic number of a graph. *Journal of Combinatorial Theory*, 1969.

13 Frnak Harary and Stephen Hedetniemi. Bounds on the chromatic and achromatic numbers of complementary graphs. *Recnet Progress in Combinatorics*, 8(2):154–161, 1970. doi:10.1016/S0021-9800(70)80072-2.

14 Pavol Hell and Donald J. Miller. Graph with given achromatic number. *Discret. Math.*, 16(3):195–207, 1976. doi:10.1016/0012-365X(76)90099-6.

15 J. Hopcroft and Mukkai Krishnamoorthy. On the harmonious coloring of graphs. *Siam Journal on Algebraic and Discrete Methods*, 4, September 1983. doi:10.1137/0604032.

16 Stasys Jukna. *Extremal Combinatorics – With Applications in Computer Science*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2011. doi:10.1007/978-3-642-17364-6.

17 Sudeshna Kolay, Ragukumar Pandurangan, Fahad Panolan, Venkatesh Raman, and Prafullkumar Tale. Harmonious coloring: Parameterized algorithms and upper bounds. *Theor. Comput. Sci.*, 772:132–142, 2019. doi:10.1016/J.TCS.2018.12.011.

18 Guy Kortsarz and Robert Krauthgamer. On Approximating the Achromatic Number. *SIAM J. Discret. Math.*, 14(3):408–422, 2001. doi:10.1137/S0895480100379579.

19 Attila Máté. A lower estimate for the achromatic number of irreducible graphs. *Discret. Math.*, 33(2):171–183, 1981. doi:10.1016/0012-365X(81)90164-3.

20 Z. Miller and D. Pritikin. The harmonious coloring number of a graph. *Discrete Mathematics*, 93(2):211–228, 1991. doi:10.1016/0012-365X(91)90257-3.

21 E. Sampathkumar. Partition graphs and coloring numbers of a graph. *Discrete Mathematics – DM*, 16:57–60, September 1976. doi:10.1016/0012-365X(76)90093-5.

22 Mihalis Yannakakis and Fanica Gavril. Edge Dominating Sets in Graphs. *SIAM J. Appl. Math.*, 38:364–372, June 1980. doi:10.1137/0138030.

New Approximate Distance Oracles and Their Applications

Avi Kadria
Bar Ilan University, Ramat Gan, Israel

Liam Roditty
Bar Ilan University, Ramat Gan, Israel

Abstract

Let $G = (V, E)$ be an undirected graph with n vertices and m edges, and let $\mu = m/n$. A *distance oracle* is a data structure designed to answer approximate distance queries, with the goal of achieving low stretch, efficient space usage, and fast query time. While much of the prior work focused on distance oracles with constant query time, this paper presents a comprehensive study of distance oracles with non-constant query time. We explore the tradeoffs between space, stretch, and query time of distance oracles in various regimes. Specifically, we consider both weighted and unweighted graphs in the regimes of stretch < 2 and stretch ≥ 2. In addition, we demonstrate several applications of our new distance oracles to the n-Pairs Shortest Paths (n-PSP) problem and the All Nodes Shortest Cycles ($ANSC$) problem. Our main contributions are:

- **Weighted graphs:** We present a new three-way trade-off between stretch, space, and query time, offering a natural extension of the classical Thorup–Zwick distance oracle [STOC'01 and JACM'05] to regimes with larger query time. Specifically, for any $0 < r < 1/2$ and integer $k \geq 1$, we construct a $(2k(1-2r)-1)$-stretch distance oracle with $\tilde{O}(m+n^{1+1/k})$ space and $\tilde{O}(\mu n^r)$ query time. This construction provides an asymptotic improvement over the classical $(2k-1)$-stretch and $O(n^{1+1/k})$-space tradeoff of Thorup and Zwick in sparse graphs, at the cost of increased query time. We also improve upon a result of Dalirrooyfard et al. [FOCS'22], who presented a $(2k-2)$-stretch distance oracle with $O(m+n^{1+1/k})$ space and $O(\mu n^{1/k})$ query time. In our oracle we reduce the stretch from $(2k-2)$ to $(2k-5)$ while preserving the same space and query time.
- **Unweighted graphs:** We present a $(2k-5, 4+2_{\text{ODD}})$-approximation[1] distance oracle with $O(n^{1+1/k})$ space and $O(n^{1/k})$ query time. This improves upon a $(2k-2, 2_{\text{ODD}})$-approximation distance oracle of Dalirrooyfard et al. [FOCS'22] while maintaining the same space and query time. We also present a distance oracle that given $u, v \in V$ returns an estimate $\hat{d}(u,v) \leq d(u,v) + 2\lceil d(u,v)/3 \rceil + 2$, using $O(n^{4/3+2\varepsilon})$ space and $O(n^{1-3\varepsilon})$ query time. To the best of our knowledge, this is the first distance oracle that simultaneously achieves a multiplicative stretch < 2, and a space complexity $O(n^{1.5-\alpha})$, for some $\alpha > 0$.
- **Applications for n-PSP and $ANSC$:** We present an $\tilde{O}(m^{1-1/(k+1)}n)$-time algorithm for the n-PSP problem, that for every input pair $\langle s_i, t_i \rangle$, where $i \in [n]$, returns an estimate $\hat{d}(s_i, t_i)$ such that $\hat{d}(s_i,t_i) \leq d(s_i,t_i) + 2\lceil d(s_i,t_i)/2k \rceil$. By allowing a small additive error, this result circumvents the conditional running time lower bound of $\Omega\left(m^{2-\frac{2}{k+1}} \cdot n^{\frac{1}{k+1}-o(1)}\right)$, established by Dalirrooyfard et al. [FOCS'22] for achieving $(1+1/k)$-stretch. Additionally, we present an $\tilde{O}(mn^{1-1/k})$-time algorithm for the $ANSC$ problem that computes, for every $u \in V$, an estimate $\hat{c}_u$ such that $\hat{c}_u \leq SC(u) + 2\lceil SC(u)/2(k-1) \rceil$, where $SC(u)$ denotes the length of the shortest cycle containing u. This improves upon the $\tilde{O}(m^{2-2/k}n^{1/k})$-time algorithm of Dalirrooyfard et al. [FOCS'22], while achieving the same approximation guarantee.

We obtain our results by developing several new techniques, among them are the *borderline vertices* technique and the *middle vertex* technique, which may be of independent interest.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Distance oracles, Fine-grained algorithms, Graph algorithms, Data structures

[1] $2_{\text{ODD}} = 2 \cdot (d(u,v) \mod 2)$.

© Avi Kadria and Liam Roditty;
licensed under Creative Commons License CC-BY 4.0
36th International Symposium on Algorithms and Computation (ISAAC 2025).
Editors: Ho-Lin Chen, Wing-Kai Hon, and Meng-Tsung Tsai; Article No. 43; pp. 43:1–43:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.43

Related Version *Full Version*: https://arxiv.org/abs/2509.00890 [27]

Funding *Liam Roditty*: Supported in part by BSF grants 2016365 and 2020356.

1 Introduction

Let $G = (V, E)$ be an undirected graph with $n = |V|$ vertices and $m = |E|$ edges with non-negative real edge weights, and let $\mu = m/n$ be the average degree. The distance $d_G(u, v)$ between u and v is the length of the shortest path[2] between u and v in G. An estimation $\hat{d}(u, v)$ of $d(u, v)$ is an α-stretch if $d(u, v) \leq \hat{d}(u, v) \leq \alpha \cdot d(u, v)$.

In their seminal work, Thorup and Zwick [36] presented a data structure for distance approximations called a *distance oracle*. A distance oracle is a data structure with $o(n^2)$ space (otherwise we can trivially store the distance matrix), that supports distance queries between vertices. For any integer $k \geq 1$, they [36] showed that in $O(kmn^{1/k})$ expected time, it is possible to preprocess a weighted undirected graph and create a distance oracle of size $O(kn^{1+1/k})$. For every $u, v \in V$ the query of the distance oracle returns a $(2k-1)$-stretch for $d(u, v)$ in $O(k)$ time. Different aspects of distance oracles, such as construction time, query time, and stretch, have been studied since their introduction, more than two decades ago. For more details see for example [10, 29, 38, 18, 19, 23, 25, 26, 21, 4, 3, 12, 13, 15].

Thorup and Zwick [36] employed Erdős' girth conjecture[3] to establish a lower bound on the space/stretch tradeoff for distance oracles. The conjecture asserts the existence of graphs with $\Omega(n^{1+1/k})$ edges and girth $\geq 2k+2$. Under this assumption, they proved that any distance oracle with stretch $t \leq 2k-1$ must use $\Omega(n^{1+1/k})$ bits on some input, for all $k \geq 1$. However, these lower bounds apply to graphs with $m = \Omega(n^{1+1/k})$ edges.

This motivates studying the $(2k-1)$-stretch/$O(n^{1+1/k})$-space tradeoff in sparser graphs, where $m = o(n^{1+1/k})$. [4] This approach has been applied in two different stretch regimes. Distance oracles with stretch ≥ 2 (see, for example, [7, 6, 9, 33, 5, 25]) and distance oracles with stretch < 2 (see, for example, [6, 8, 21, 28, 31]).

1.1 Distance oracles with stretch ≥ 2

Our main result for stretch ≥ 2 establishes a three-way tradeoff among the key parameters of distance oracles: stretch, space, and query time. This new tradeoff improves upon previously known results for specific values of stretch, space, and query time. We prove the following:

▶ **Theorem 1.** *Let $k \geq 5$ and let $0 < c < \frac{k}{2} - 1$ be an integer. There is an $\tilde{O}(m + n^{1+1/k})$ space distance oracle that given two query vertices $u, v \in V$ computes in $O(\mu n^{c/k})$-time a distance estimation $\hat{d}(u, v)$ that satisfies $d(u, v) \leq \hat{d}(u, v) \leq (2k - 1 - 4c) \cdot d(u, v)$. The distance oracle is constructed in $\tilde{O}(mn^{\frac{c+2}{k}})$ expected time.*

An interesting tradeoff that follows from Theorem 1 is a better space/stretch tradeoff than the classical Thorup and Zwick [36] $(2k-1)$-stretch / $\tilde{O}(n^{1+1/k})$ space tradeoff, at a cost of a larger query time. Specifically, by setting $r = \frac{c}{k}$, we get a $(2k(1-2r)-1)$-stretch

[2] We omit G when it is clear from the context.
[3] The girth of a graph is the length of its shortest cycle.
[4] We remark that a distance oracle in such graphs must use $\Omega(m)$ space, because of similar arguments to the lower bound of [36].

distance oracle that uses $\tilde{O}(m+n^{1+1/k})$ space, at the price of $\tilde{O}(\mu n^r)$ query time. By setting $c=1$ in Theorem 1, we improve upon a recent result of Dalirrooyfard, Jin, V. Williams, and Wein [22]. They presented a $(2k-2)$-stretch distance oracle that uses $\tilde{O}(m+n^{1+1/k})$ space and $\tilde{O}(\mu n^{1/k})$ query time.[5] We improve the stretch from $(2k-2)$ to $(2k-5)$ while using the same space and query time.

To obtain Theorem 1, we develop a new technique called the *borderline vertices* technique. Given $u,v \in V$, the borderline vertices τ_u and τ_v are two vertices on the shortest path between u and v that satisfy special properties that allow us to better exploit the structure of the Thorup and Zwick distance oracle. Using the borderline vertices technique, we also manage to achieve the following distance oracle, which improves upon the construction time of Theorem 1 at the cost of increasing the stretch.

▶ **Theorem 2.** *Let $k \geq 4$ and let $0 < c < \frac{k}{2} - 1$ be an integer. There is an $\tilde{O}(m+n^{1+1/k})$ space distance oracle that given two query vertices $u,v \in V$ computes in $\tilde{O}(\mu n^{c/k})$-time a distance estimation $\hat{d}(u,v)$ that satisfies $d(u,v) \leq \hat{d}(u,v) \leq (2k-4c)\cdot d(u,v)$. The distance oracle is constructed in $\tilde{O}(mn^{\frac{c+1}{k}})$ expected time.*

Agarwal, Godfrey, and Har-Peled [9] considered also small stretches, and presented a 2-stretch distance oracle that uses $\tilde{O}(m+n^{3/2})$ space and has $\tilde{O}(\mu n^{1/2})$ query time, that is constructed in $\tilde{O}(mn^{1/2})$ time, and a 3-stretch distance oracle that uses $\tilde{O}(m+n^{4/3})$ space and has $\tilde{O}(\mu n^{1/3})$ query time, and is constructed in $\tilde{O}(mn^{2/3})$ time. These results suggest that the following general stretch/space/query time tradeoff may exist for every k.

▶ **Problem 1.** *For which values of k it is possible to construct a k-stretch distance oracle that uses $\tilde{O}(m+n^{1+1/k})$ space and has $\tilde{O}(\mu n^{1/k})$ query time.*

Agarwal, Godfrey, and Har-Peled [9] solved Problem 1 for $k=2,3$. By setting $c=1$ in Theorem 2 and Theorem 1, respectively, we obtain a 4-stretch distance oracle that uses $\tilde{O}(m+n^{5/4})$ space and has $\tilde{O}(\mu n^{1/4})$ query time, and a 5-stretch distance oracle that uses $\tilde{O}(m+n^{6/5})$ space and has $\tilde{O}(\mu n^{1/5})$ query time. Thus, we solve Problem 1 for $k=4,5$.

In addition, using the borderline vertices technique we obtain $(2k-3)$-stretch distance oracle that uses $\tilde{O}(m+n^{1+1/k})$ space, has $\tilde{O}(\mu n^{1/k})$ query time, and is constructed in $\tilde{O}(mn^{1/k})$ time. For $k=3$ this improves the construction time of [9] for a 3-stretch distance oracle from $\tilde{O}(mn^{2/3})$ to $\tilde{O}(mn^{1/3})$. Our results for weighted graphs are summarized in Table 2.

Next, we turn our focus to *unweighted* graphs. An estimation $\hat{d}(u,v)$ of $d(u,v)$ is an (α,β)-approximation if $d(u,v) \leq \hat{d}(u,v) \leq \alpha d(u,v)+\beta$. (α,β)-approximations were extensively studied in the context of distance oracles, graph spanners, and emulators. (For more details see for example [24, 37, 11, 17, 16, 30, 1, 2, 9, 7, 13]).

From the girth conjecture, it follows that in unweighted graphs, for every (α,β)-distance oracle that uses $\tilde{O}(n^{1+1/k})$ space, it must hold that $\alpha+\beta \geq 2k-1$. However, this inequality must hold only for adjacent vertices. For non-adjacent vertex pairs, we prove the following simple lower bound.

▶ **Theorem 3.** *Assuming the Erdős girth conjecture, any distance oracle that uses $o(n^{1+1/k})$ space must have an input graph $G=(V,E)$ and two vertices $u,v \in V$ such that $d(u,v)=2$ and $\hat{d}(u,v) \geq 2k+2$.*

[5] We remark that even more recently, Chechik, Hoch, and Lifshitz [20] presented a $(2k-3)$-stretch n-PSP algorithm, however, they did not present a new distance oracle.

A *spanner* is a subgraph $H \subseteq G$ that approximates distances *without* supporting distance queries. Baswana, Kavitha, Mehlhorn, and Pettie [11] and Parter [30] presented a k-spanner for every $u, v \in V$ such that $d(u, v) = 2$, and matched the lower bound of Theorem 3 for spanners. An interesting question is whether this lower bound can be matched by distance oracles with efficient distance queries as well. Thus, we formulate the following problem.

▶ **Problem 2.** *For which values of k it is possible to construct a distance oracle with $\tilde{O}(n^{1+1/k})$ space that uses $\tilde{O}(n^{1/k})$ query time, such that $d(u, v) \leq \hat{d}(u, v) \leq k \cdot d(u, v)$, for every $u, v \in V$ that satisfy $d(u, v) = 2$.*

Agarwal, Godfrey, and Har-Peled [9] presented for unweighted graphs a $(2, 1)$-approximation distance oracle that uses $\tilde{O}(n^{3/2})$ space, has $\tilde{O}(n^{1/2})$ query time, and is constructed in $\tilde{O}(mn^{1/2})$ time. They also presented a $(3, 2)$-approximation distance oracle that uses $\tilde{O}(n^{4/3})$ space, has $\tilde{O}(n^{1/3})$ query time, that is constructed in $\tilde{O}(mn^{2/3})$ time.

By analyzing more carefully the $(2, 1)$-approximation distance oracle of [9] it is possible to show that it is actually a $(2, 1_{\text{ODD}})$-approximation[6] distance oracle, and therefore solves Problem 2 for $k = 2$.[7] In the following two theorems, we show that it is possible to solve Problem 2 for $k = 3$ and $k = 4$ as well.

▶ **Theorem 4.** *There is a $\tilde{O}(n^{4/3})$ space distance oracle that given two query vertices $u, v \in V$ computes in $\tilde{O}(n^{1/3})$-time a distance estimation $\hat{d}(u, v)$ that satisfies $d(u, v) \leq \hat{d}(u, v) \leq 3d(u, v) + 2_{\text{ODD}}$. The distance oracle is constructed in $\tilde{O}(mn^{1/3})$ expected time.*

▶ **Theorem 5.** *There is a $\tilde{O}(n^{5/4})$ space distance oracle that given two query vertices $u, v \in V$ computes in $\tilde{O}(n^{1/4})$-time a distance estimation $\hat{d}(u, v)$ that satisfies $d(u, v) \leq \hat{d}(u, v) \leq 4d(u, v) + 3_{\text{ODD}}$. The distance oracle is constructed in $\tilde{O}(mn^{1/2})$ expected time.*

In Theorem 4 we improve the distance oracle of [9] from more than a decade ago. In particular we improve the construction time from $\tilde{O}(mn^{2/3})$ to $\tilde{O}(mn^{1/3})$, and the approximation from $(3, 2)$ to $(3, 2_{\text{ODD}})$. Dalirrooyfard, Jin, V. Williams, and Wein [22] presented a $(2k - 2, 2_{\text{ODD}})$-stretch distance oracle that uses $\tilde{O}(n^{1+1/k})$ space and $\tilde{O}(n^{1/k})$ query time. By setting $k = 4$ in the $(2k-2, 2_{\text{ODD}})$-stretch distance oracle of [22] they obtain a $(6, 2_{\text{ODD}})$-stretch distance oracle. In Theorem 5 we improve the approximation from $(6, 2_{\text{ODD}})$ to $(4, 3_{\text{ODD}})$ while maintaining the same space and query time.

To obtain our new distance oracles for unweighted graphs, presented in Theorem 4 and Theorem 5, we develop a new technique for using the middle vertex in the path, called the *middle vertex* technique. Let $u, v \in V$, the middle vertex τ is a vertex on the shortest path between u and v that satisfies $d(u, \tau) = d(u, v)/2 + 0.5_{\text{ODD}}$ and $d(v, \tau) = d(u, v)/2 - 0.5_{\text{ODD}}$. As with the borderline vertices technique for *sparse weighted* graphs, the middle vertex technique enables us to better exploit the structure of the distance oracles of Thorup and Zwick in *dense unweighted* graphs.

By combining the middle vertex technique with the borderline vertices technique, we manage to achieve two more distance oracles for dense unweighted graphs. The first distance oracle improves the approximation of the distance oracle of Theorem 5 from $(4, 3_{\text{ODD}})$ to $(3, 2 + 2_{\text{ODD}})$ at the cost of increasing the query time to $\tilde{O}(n^{1/2})$.

---

[6] x_{ODD} is defined as $x \cdot 1_{\text{ODD}}$, where 1_{ODD} is $d(u, v) \mod 2$

[7] In their work, the additive error comes from the fact that if $B(u) \cap B(v) = \emptyset$ then $\min(h(u), h(v)) \leq d(u, v)/2 + 1$. However, this can be improved by showing that if $B(u) \cap B(v) = \emptyset$ then $\min(h(u), h(v)) \leq d(u, v)/2 + 1_{\text{ODD}}$.

▶ **Theorem 6.** *There is a $\tilde{O}(n^{5/4})$ space distance oracle that given two query vertices $u, v \in V$ computes in $\tilde{O}(n^{1/2})$-time a distance estimation $\hat{d}(u,v)$ that satisfies $d(u,v) \leq \hat{d}(u,v) \leq 3d(u,v) + 2 + 2_{\text{ODD}}$. The distance oracle is constructed in $\tilde{O}(mn^{1/2})$ expected time.*

The second distance oracle improves the $(2k-2, 2_{\text{ODD}})$-approximation of [22] to $(2k-5, 4+2_{\text{ODD}})$-approximation, while using the same space and query time.

▶ **Theorem 7.** *Let $k \geq 5$. There is an $\tilde{O}(n^{1+1/k})$ space distance oracle that given two query vertices $u, v \in V$ computes in $\tilde{O}(n^{1/k})$-time a distance estimation $\hat{d}(u,v)$ that satisfies $d(u,v) \leq \hat{d}(u,v) \leq (2k-5)d(u,v) + 4 + 2_{\text{ODD}}$. The distance oracle is constructed in $\tilde{O}(mn^{\frac{3}{k}})$ expected time.*

Our results for unweighted graphs are summarized in Table 1.

1.2 Distance oracles with stretch < 2

For distance oracles with stretch < 2 in weighted graphs, we present the following new three-way tradeoff between: stretch, space, and query time. We prove:

▶ **Theorem 8.** *Let $0 < c < 1/3$ be a real constant. There is a $\tilde{O}(m + n^{2-2c})$-space distance oracle that given two query vertices $u, v \in V$ computes in $\tilde{O}(\mu^t n^{tc})$-time a distance estimation $\hat{d}(u,v)$ that satisfies $d(u,v) \leq \hat{d}(u,v) \leq (1 + 2/t)d(u,v)$. The distance oracle is constructed in $\tilde{O}(mn^{1-c})$ time.*

Using this tradeoff, we obtain the first distance oracle with stretch < 2, sublinear query time, and $\tilde{O}(m + n^{1.5-\alpha})$ space, for $\alpha > 0$. In particular, by setting $t = 3$ and $c = 1/3 - \varepsilon$, we obtain a 5/3-stretch distance oracle with $\tilde{O}(m + n^{4/3+2\varepsilon})$-space and $O(\mu^3 n^{1-3\varepsilon})$-query time. For comparison, Agarwal [6] presented a distance oracle with stretch $1 + \frac{1}{t+0.5}$, using $\tilde{O}(m + n^{2-c})$ space and $\tilde{O}(\mu^t n^{(t+1)c})$ query time. Setting $t = 1$ and $c = 1/2 - \varepsilon$ in the construction of [6] yields a 5/3-stretch oracle with $\tilde{O}(m + n^{1.5+\varepsilon})$-space and $\tilde{O}(\mu n^{0.5-\varepsilon})$-query time. (See Table 2.)

For unweighted graphs, we present two new tradeoffs. The first improves upon the tradeoff of the distance oracle of Bilò, Chechik, Choudhary, Cohen, Friedrich, and Schirneck [13], while preserving the same space and query time. Specifically, they presented a distance oracle for unweighted graphs that uses $\tilde{O}(n^{2-c})$ space, has $\tilde{O}(n^{tc})$ query time, and returns an estimate $\hat{d}(u,v) \leq (1 + 1/t)d(u,v) + 2$. In the following theorem, we slightly improve the stretch bound to $\hat{d}(u,v) \leq d(u,v) + 2\left\lceil \frac{d(u,v)}{2t} \right\rceil$, while using the same space and query time.

▶ **Theorem 9.** *Let $t \geq 1$ be an integer and let $0 < c < 1/2$ be a real constant. There is a $\tilde{O}(n^{2-c})$-space distance oracle that given two query vertices $u, v \in V$ computes in $\tilde{O}(n^{c \cdot t})$-time a distance estimation $\hat{d}(u,v)$ that satisfies $d(u,v) \leq \hat{d}(u,v) \leq d(u,v) + 2\left\lceil \frac{d(u,v)}{2t} \right\rceil$. The distance oracle is constructed in $\tilde{O}(mn^{1-c})$ time.*

Next, by adapting methods of Theorem 8 to unweighted graphs, we obtain another tradeoff that uses less space than the tradeoff of Theorem 9 at the cost of a larger query time. We prove:

▶ **Theorem 10.** *Let $t \geq 1$. Let $0 < c < 1/3$ be a real constant. There is a $\tilde{O}(n^{2-2c})$-space distance oracle that given two query vertices $u, v \in V$ computes in $\tilde{O}(n^{tc})$-time a distance estimation $\hat{d}(u,v)$ that satisfies $d(u,v) \leq \hat{d}(u,v) \leq d(u,v) + 2\lceil d(u,v)/t \rceil + 2$. The distance oracle is constructed in $\tilde{O}(mn^{1-c})$ time.*

Table 1 Distance oracles in unweighted graphs. x_{ODD} is $x \cdot (d(u,v) \mod 2)$.

Space	Time	Old Approximation	New Approximation
$\tilde{O}(kn^{1+1/k})$	$\tilde{O}(n^{1/k})$	$(2k-2, 2_{\text{ODD}})$ [22]	$(2k-5, 4+2_{\text{ODD}})$ (Th7)
$\tilde{O}(n^{4/3})$	$\tilde{O}(n^{1/3})$	$(3,2)$ [7]	$(3, 2_{\text{ODD}})$ (Th4)
$\tilde{O}(n^{5/4})$	$\tilde{O}(n^{1/4})$	$(6, 2_{\text{ODD}})$ [22]	$(4, 3_{\text{ODD}})$ (Th5)
$\tilde{O}(n^{5/4})$	$\tilde{O}(n^{1/2})$	–	$(3, 2+2_{\text{ODD}})$ (Th6)
$\tilde{O}(n^{6/5})$	$\tilde{O}(n^{1/5})$	$(8, 2_{\text{ODD}})$ [22]	$(5, 4+2_{\text{ODD}})$
$\tilde{O}(n^{2-c})$	$\tilde{O}(n^{tc})$	$\delta + \delta/t + 2$ [13]	$\delta + 2\lceil \delta/2t \rceil$ (Th9)
$\tilde{O}(n^{3/2+\epsilon})$	$\tilde{O}(n^{1-2\epsilon})$	$1.5\delta + 2$ [13]	$\delta + 2\lceil \delta/4 \rceil$ (Th9)
$\tilde{O}(n^{2-2c)}$	$\tilde{O}(n^{tc})$	–	$\delta + 2\lceil \delta/t \rceil + 2$ (Th10)
$\tilde{O}(n^{4/3+2\epsilon})$	$\tilde{O}(n^{1-3\epsilon})$	–	$\delta + 2\lceil \delta/3 \rceil + 2$ (Th10)

In particular, using Theorem 10 we obtain the first distance oracle with $o(n^{1.5-\alpha})$-space, for $\alpha > 0$, that achieves a multiplicative error strictly better than 2. Specifically, by setting $t = 3$ and $c = 1/3 - \varepsilon$, we get a distance oracle with $\tilde{O}(n^{4/3+2\varepsilon})$ space and $\tilde{O}(n^{1-3\varepsilon})$ query time. These results are summarized in Table 1.

1.3 Applications for n-PSP and $ANSC$

Recently, Dalirrooyfard, Jin, V. Williams, and Wein [22] (followed by Chechik, Hoch and Lifshitz [20]) studied two fundamental problems in graphs, the n-pairs shortest paths (n-PSP) problem and the all-nodes shortest cycles ($ANSC$) problem. The n-PSP problem is defined as follows. Given a set of vertex pairs (s_i, t_i), for $1 \le i \le O(n)$, compute the distance $d(s_i, t_i)$. The $ANSC$ problem is defined as follows. Compute the length of the shortest simple cycle that contains v, for every $v \in V$.

A straightforward approach for solving the n-PSP problem is to first construct a distance oracle and then query it for every pair in the input set. The total runtime of this approach is the sum of the construction time and n times the query time of the distance oracle. While this method naturally yields an n-PSP algorithm, a distance oracle must satisfy additional requirements that an n-PSP algorithm does not. In particular, a distance oracle must use limited space and support queries between any of the $\Omega(n^2)$ pairs of vertices.

Using our new distance oracles, we obtain improved time/stretch tradeoffs for both the n-PSP problem and the $ANSC$-problem, improving some of the results of [22, 20].

In [22], a lower bound based on the combinatorial $4k$ clique hypothesis for the n-PSP problem is presented. They showed that $\Omega(m^{2-\frac{2}{k+1}} \cdot n^{\frac{1}{k+1}-o(1)})$ time is required to achieve $(1 + 1/k - \varepsilon)$ stretch. As mentioned in [22] using the $(1 + 1/k)$-distance oracle of [6] it is possible to solve n-PSP with $(1 + 1/k)$-stretch in $O(m^{2-\frac{2}{k+1}} n^{\frac{1}{k+1}})$ time, which almost matches the lower bound of [22]. In this paper, we show that it is possible to significantly improve the running time of [6] and to bypass the lower bound of [22] at the cost of a small additive error (of up to 2). We obtain an $O(m^{1-\frac{1}{k+1}} n)$ time algorithm that returns $\hat{d}(s_i, t_i) \le d(s_i, t_i) + 2\left\lceil \frac{d(s_i,t_i)}{2k} \right\rceil \le (1 + 1/k)d(u, v) + 2$, as presented in the following theorem.

▶ **Theorem 11.** *Let $k \ge 2$. Given an unweighted graph G and vertex pairs (s_i, t_i) for $1 \le i \le O(n)$, there is an algorithm for $n-PSP$ that computes an estimation $\hat{d}(s_i, t_i)$ such that $d(s_i, t_i) \le \hat{d}(s_i, t_i) \le d(s_i, t_i) + 2\left\lceil \frac{d(s_i,t_i)}{2k} \right\rceil$ in $\tilde{O}(m^{1-\frac{1}{k+1}} n)$ time.*

In [22] they presented an $O(m+n^{2/k})$ time n-PSP algorithm that returns $(2k-2)(2k-1)$-stretch. In the following theorem we improve the stretch from $(2k-1)(2k-2)$ to $(2k-1)(2k-3)$ while using the same running time.

▶ **Theorem 12.** *Let $k \geq 3$. Given a weighted graph G and vertex pairs (s_i, t_i) for $1 \leq i \leq O(n)$, there is an algorithm for $n - PSP$ that computes an estimation $\hat{d}(s_i, t_i)$ such that $d(s_i, t_i) \leq \hat{d}(s_i, t_i) \leq (2k-1)(2k-3)$ in $\tilde{O}(m + n^{2/k})$ time.*

For the $ANSC$ problem, [22] presented an $O(m^{2-2/k}n^{1/k})$ running time algorithm that for every $u \in V$ returns estimate $\hat{c}(u)$ such that $\hat{c}_u \leq SC(u) + 2\left\lceil \frac{SC(u)}{2(k-1)} \right\rceil$, where $SC(u)$ is the length of the shortest simple cycle that contains u. In the following theorem, we improve the running time from $O(m^{2-2/k}n^{1/k})$ of [22] to $\tilde{O}(mn^{1-\frac{1}{k}})$.

▶ **Theorem 13.** *Given an undirected unweighted graph G, let k be a positive integer. There is a randomized algorithm for $ANSC$ that computes for every $u \in V$ an estimation $\hat{c}_u$ such that $SC(u) \leq \hat{c}_u \leq 1 + 2\left\lceil \frac{SC(u)}{2(k-1)} \right\rceil$ in $\tilde{O}(mn^{1-\frac{1}{k}})$-time.*

In addition, we present the first, to the best of our knowledge, stretch < 2 algorithm for the $ANSC$ problem in *weighted graphs*. Specifically, we present an $O(m^{2-1/k})$ time algorithm for weighted graphs that for every $u \in V$ returns an estimate $\hat{c}$ such that $\hat{c}_u \leq 1 + \frac{1}{k-1}SC(u)$, as presented in the following theorem.

▶ **Theorem 14.** *Given an undirected weighted graph G, let k be a positive integer. There is a randomized algorithm for $ANSC$ that computes for every $u \in V$ an estimation $\hat{c}_u$ such that $SC(u) \leq \hat{c}_u \leq (1 + \frac{1}{k-1})SC(u)$ in $\tilde{O}(m^{2-\frac{1}{k}})$-time.*

Our results for the $ANSC$ problem are summarized in Table 4.

The rest of this paper is organized as follows. In the next section, we present some necessary preliminaries. In Section 3 we present a technical overview of our main techniques and some of our new distance oracles. In the full version [27] of the paper, we present the full proofs of our techniques, distance oracles, applications, and lower bound.

2 Preliminaries

In this section, we present several definitions and existing tools that we use to develop our new techniques and ideas which we then apply in order to obtain new distance oracles. Let $G = (V, E)$ be an undirected graph with $n = |V|$ vertices and $m = |E|$ edges. Throughout the paper, we consider both unweighted graphs and weighted graphs with non-negative real edge weights.

Let $u, v \in V$. The distance $d(u, v)$ between u and v is the length of a shortest path between u and v. Let $P(u, v)$ be the shortest path between u and v. Let $N(u)$ be the vertices that are neighbors of u and let $deg(u) = |N(u)|$ be the degree of u. Let $X \subseteq V$. Let $N(X)$ be the vertices that are neighbors of some vertex $u \in X$, i.e, $N(X) = \{w \in V | \exists x \in X : (w, x) \in E\}$. The distance $d(u, X)$ between u and X is the distance between u and the closest vertex to u from X, that is, $d(u, X) = \min_{x \in X}(d(u, x))$. Let $p(u, X) = \arg\min_{x \in X}(d(u, x))$ (ties are broken in favor of the vertex with a smaller identifier).

Next, we define bunches and clusters as in [36]. Let $u \in V$ and let $X, Y \subseteq V$. Let $B(u, X, Y) = \{v \in X \mid d(u, v) < d(u, Y)\}$ be the bunch of u with respect to X and Y. Let $C(u, Y) = \{v \in V \mid d(u, v) < d(v, Y)\}$ be the cluster of u with respect to Y.

The starting point in many algorithms and data structures for distance approximation, and in particular in Thorup and Zwick[36]'s distance oracles, is a hierarchy of vertex sets $A_0, A_1, \ldots, A_k$, where $A_0 = V$, $A_k = \emptyset$ and for $0 \leq i \leq k-1$, $A_{i+1} \subseteq A_i$ and $|A_i| = n^{1-i/k}$. For every $u \in V$, let $p_i(u) = p(u, A_i)$ and let $h_i(u) = d(u, A_i)$. Using this hierarchy, Thorup and Zwick defined k bunches for every vertex $u \in V$ as follows: For every $0 \leq i \leq k-1$, the bunch $B_i(u)$ is defined as $B_i(u) = B(u, A_i, A_{i+1})$. They also defined a cluster for every

Algorithm 1 `Intersection`(u, w, U, W).

```
1 if U ∩ W ≠ ∅ then
2     return min_{x∈U∩W} d'(u, x) + d'(x, w);
3 return ∞;
```

vertex $w \in A_i \setminus A_{i+1}$ as follows: $C(w) = C(w, A_{i+1})$. Throughout the paper when we save $B_i(u)$, for every $u \in V$ and $0 \leq i \leq k-1$ (or $C(u) = C(u, A_{i+1})$, for every $u \in A_i \setminus A_{i+1}$), we mean that we can check whether $x \in B_i(u)$ (or $x \in C(u)$) in constant time and if $x \in B_i(u)$ (or $x \in C(u)$) we can retrieve $d(u, x)$ in constant time as well.

The simplest instance of this hierarchy is when $k = 2$ and the hierarchy is composed of the sets A_0, A_1, A_2, where $A_0 = V$, $A_1 \subset V$, and $A_2 = \emptyset$. Throughout this paper, in such a case, we will omit subscripts and will refer to A_1 simply as A. Moreover, since in this case $B_0(u) = B(u, V, A)$ and $B_1(u) = B(u, A, \emptyset) = A$, for every $u \in V$, we use $B(u)$ to denote $B_0(u)$ and A to denote $B_1(u)$ and since we have only $p_0(u)$ and $p_1(u)$, where $p_0(u)$ is u, we use $p(u)$ to denote $p_1(u)$ and $h(u)$ to denote $h_1(u)$.

Thorup and Zwick [36] construct the sets $A_1, \ldots, A_{k-1}$, by adding to A_{i+1}, where $i \in [0, k-2]$, every vertex of A_i, independently at random with probability p. Constructing the sets in such a way allows Thorup and Zwick [36] to prove the following:

▶ **Lemma 15** ([36]). *Given an integer parameter $k \geq 2$, we can compute in $\tilde{O}(n)$ time sets $A_1, \ldots, A_{k-1}$, such that $|A_i| = O(n^{1-i/k})$ for every $i \in [1, k-1]$ with high probability, and for every $i \in [0, k-1]$ the size of $B_i(u)$ is $O(n^{1/k})$, with high probability (w.h.p). The cost of computing $B_i(u)$, for every $u \in V$, is $\tilde{O}(mn^{1/k})$ expected time.*

▶ **Lemma 16** ([35]). *Given a parameter p, we can compute a set A of size $\tilde{O}(np)$ in $\tilde{O}(mp^{-1})$ expected time such that, $|C(w, A)| = O(1/p)$, for every vertex $w \in V \setminus A$, and $|B(v, V, A)| = O(1/p)$ for every $v \in V$.*

We remark that both Lemma 15 and Lemma 16 have also slower deterministic constructions [34, 35].

Next, we present procedure `Intersection`. The input to `Intersection` is two vertices $u, w \in V$ and two sets $U, W \subseteq V$, such that, $u \in U$ and $w \in W$. For every $u' \in U$ and every $w' \in W$, we assume that distance estimations $d'(u, u')$ and $d'(w, w')$ are known. The output of `Intersection` is $\min_{x \in U \cap W} d'(u, x) + d'(w, x)$, if $U \cap W \neq \emptyset$ and ∞, otherwise. (See Algorithm 1.) In most of our uses of `Intersection` we have $d'(u, u') = d(u, u')$ and $d'(w, w') = d(w, w')$. When this is not the case, we explicitly describe d' and its relation to d. The following lemma regarding the running time of `Intersection` is straightforward.

▶ **Lemma 17.** *`Intersection`(u, w, U, W) runs in $O(\min(|U|, |W|))$.*

Proof. Assume, without loss of generality (wlog), that $|U| \geq |W|$. By checking for every vertex $y \in W$ in $O(1)$ time whether $y \in U$ we compute $U \cap W$. For each vertex $x \in U \cap W$, we compute in $O(1)$ time the value $d'(u, x) + d'(x, w)$. Thus, the total running time is $\min(|U|, |W|) \cdot O(1) = O(\min(|U|, |W|))$, as required. ◀

The following property of `Intersection`(u, w, U, W) is the main feature of `Intersection`, and will be used throughout the paper.

▶ **Property 18.** *Let $P = P(u, w)$. If $P \cap (U \cap W) \neq \emptyset$, and there exists $x \in P \cap (U \cap W)$ such that $d'(u, x) = d(u, x)$ and $d'(x, v) = d(x, v)$ then `Intersection`(u, w, U, W) returns $d(u, w)$.*

Algorithm 2 `TZQuery`(u, v).

```
for i ∈ [k] do
    if p_{i-1}(u) ∈ B_i(v) then return h_{i-1}(u) + d(p_{i-1}(u), v);
    (u, v) ← (v, u)
```

Let `TZQuery` be the query of the distance oracle presented by Thorup and Zwick [36]. We let `MTZQuery`(u, v) be $\min(\texttt{TZQuery}(u, v), \texttt{TZQuery}(v, u))$. Pseudocode for `TZQuery` exists in Algorithm 2.

The following lemma appears explicitly in [36] regarding the correctness and running time of `MTZQuery`.

▶ **Lemma 19** ([36]). *The running time of `MTZQuery` is $O(k)$ and it holds that*

$$\texttt{MTZQuery}(u, v) \leq (2k-1)d(u, v)$$

The next lemma follows from [36], and we prove it here for completeness.

▶ **Lemma 20.** *Let $\hat{d}(u, v)$ be `MTZQuery`(u, v). For every integer $1 \leq i \leq k$, one of the following holds.*

- $\min(h_i(u), h_i(v)) \leq \min(h_{i-1}(u), h_{i-1}(v)) + d(u, v)$
- $\hat{d}(u, v) \leq 2\min(h_{i-1}(u), h_{i-1}(v)) + d(u, v)$

Proof. Wlog, assume that $h_{i-1}(u) \leq h_{i-1}(v)$. We divide the proof into two cases. The case that $p_{i-1}(u) \in B_i(v)$ and the case that $p_{i-1}(u) \notin B_i(v)$. Consider the case that $p_{i-1}(u) \in B_i(v)$. In this case the value of $d(p_{i-1}(u), v)$ is saved in the distance oracle. From the triangle inequality it follows that $d(p_{i-1}(u), v) \leq h_{i-1}(u) + d(u, v)$. Therefore, we have that $\hat{d}(u, v) \leq h_{i-1}(u) + d(p_{i-1}(u), v) \leq 2h_{i-1}(u) + d(u, v)$. Since $h_{i-1}(u) \leq h_{i-1}(v)$ we get that $\hat{d}(u, v) \leq 2\min(h_{i-1}(u), h_{i-1}(v)) + d(u, v)$, as required.

Consider now the case that $p_{i-1}(u) \notin B_i(v)$. From the definition of $B_i(v)$ it follows that $h_i(v) \leq d(v, p_{i-1}(u)) \leq d(u, v) + h_{i-1}(u)$. Since $h_{i-1}(u) \leq h_{i-1}(v)$, we get that $h_i(v) \leq \min(h_{i-1}(u), h_{i-1}(v)) + d(u, v)$, and thus $\min(h_i(u), h_i(v)) \leq \min(h_{i-1}(u), h_{i-1}(v)) + d(u, v)$, as required. ◀

The following property follows by applying Lemma 20 inductively.

▶ **Property 21.** *Let $1 \leq i \leq k$ be an integer, and let $1 \leq j < i$. Then either:*

- $\min(h_i(u), h_i(v)) \leq \min(h_{i-1}(u), h_{i-1}(v)) + d(u, v)$
- $\hat{d}(u, v) \leq 2\min(h_j(u), h_j(v)) + (i - j)d(u, v)$

We remark that all our distance oracles return an estimation that is the length of a path in G and therefore $d(u, v) \leq \hat{d}(u, v)$.

3 Technical overview

In the classic distance oracle of Thorup and Zwick [36], it is possible to bound $h_i(u)$ with $d(u, v) + h_{i-1}(v)$, for every $1 \leq i \leq k-1$. The key to improving the $(2k-1)$-stretch of the distance oracle lies in tightening the bound on $h_i(u)$. Previous work (see, for example, [9, 32, 5, 8, 6]) improved the bound on $h_1(u)$ from $d(u, v)$ to $d(u, v)/2$ by exploiting the following structural property of $B_0(\cdot)$: if $P(u, v) \not\subseteq B_0(u) \cup B_0(v)$, then $h_1(u) + h_1(v) \leq d(u, v) + 1_{\text{ODD}}$.

In this paper, we develop two new techniques that exploit structural properties of $C(\cdot)$ to construct a distance oracle with improved stretch guarantees. These techniques enable us to bound not only $h_1(u)$ by $d(u,v)/2$, but also $h_2(u)$ by $d(u,v)$, thereby improving overall stretch. We begin by presenting the borderline vertices technique, which applies to both weighted and unweighted graphs.

3.1 The borderline vertices technique

Let $C^*(u) = C(u) \cup N(C(u))$ be an augmented cluster. Roughly speaking, using the borderline vertices technique we show that if $P(u,v) \not\subseteq C^*(u,A_c) \cap C^*(v,A_c)$ then $\max(h_{c+1}(u), h_{c+1}(v)) \leq d(u,v)$. In particular, by setting $c=1$ we get an improvement over the result of [9], showing not only that $h_1(u) \leq d(u,v)/2$ but also that $h_2(u) \leq d(u,v)$.

Let $P = P(u,v)$ be a shortest path between u and v. The borderline vertex $\tau_u(P)$ of $C^*(u)$ in P is the farthest vertex from u in $C^*(u) \cap P$. Similarly, the borderline vertex $\tau_v(P)$ of $C^*(u)$ in P is the farthest vertex from v in $C^*(v) \cap P$. Obviously, a distance oracle cannot store $\tau_u(P)$ for every shortest path P, as this would require $\Theta(n^2)$ space. We overcome this limitation by using $\tilde{O}(\mu \cdot |C(u)|)$ query time to iterate over all vertices in the augmented cluster $C^*(u)$, and in particular $\tau_u = \tau_u(P)$.

To obtain our improved bound on $h_2(v)$, we analyze two different cases regarding $p_1(\tau_u)$. The case that $p_1(\tau_u) \in B_1(v)$ and the case that $p_1(\tau_u) \notin B_1(v)$. If $p_1(\tau_u) \in B_1(v)$, then the query algorithm can return as an estimation $\hat{d}(u,v) = u \to \tau_u \to p_1(\tau_u) \to v$, where $x \to y \to z \to w$ is $d(x,y)+d(y,z)+d(z,w)$. In this case, since $\tau_u \notin C(u,A_1)$ (as it is the farthest vertex from u in $C^*(u)$) we get that $h_1(\tau_u) \leq d(u,\tau_u) \leq d(u,v)$, and therefore $\hat{d}(u,v) \leq 3d(u,v)$.

Otherwise, if $p_1(\tau_u) \notin B_1(v)$ then $h_2(v) \leq d(v,p_1(\tau_u)) \leq d(v,\tau_u) + h_1(\tau_u)$. Since $h_1(\tau_u) \leq d(u,\tau_u)$ we get that $h_2(v) \leq d(v,\tau_u) + d(u,\tau_u) = d(u,v)$, as wanted. (See Figure 1). Using symmetric arguments for τ_v, we can also get that $h_2(u) \leq d(u,v)$.

In unweighted graphs, we can avoid the μ factor from the query time at the cost of bounding $h_2(u)$ and $h_2(v)$ by $d(u,v)+2$ rather than $d(u,v)$. The difference is that we let $\tau'_u(P)$ be defined as the farthest vertex from u in $C(u) \cap P$ (unlike $\tau_u(P)$ which considers $C^*(u) \cap P$). Next, we provide an overview of our main three-way tradeoff that uses the borderline vertices technique.

3.2 $(2k-1-4c)$-stretch distance oracle

Let $0 < c < k/2 - 1$ be an integer, let $V = A_0, A_c, A_{c+1}, \ldots, A_k = \emptyset$, such that $|A_i| = n^{1-i/k}$. The storage of the distance oracle saves: the graph G, $d(s_1,s_2)$, for every $s_1 \in A_{c+1}, s_2 \in A_{k-c-2}$ and $B(u) = \cup_{i \in \{0 \cup [c,k]\}} B_i(u)$ for every $u \in V$. Where $B_0(u) = B(u, V, A_c)$.

The query works as follows. First, we set $\hat{d}(u,v)$ to $\texttt{Intersection}(u, v, C^*(u,A_c), C^*(v,A_c))$. Then, we iterate over every $u' \in C^*(u,A_c)$ (to iterate over τ_u), and update $\hat{d}(u,v)$ be $\min(\hat{d}(u,v), d(u,u') + \texttt{MTZQuery}(u',v))$. Similarly, for every $v' \in C^*(v,A_c)$ we update $\hat{d}(u,v)$ to $\min(\hat{d}(u,v), d(v,v') + \texttt{MTZQuery}(v',u))$. Finally, the algorithm returns $\min(\hat{d}(u,v), u \to p_{c+1}(u) \to p_{k-c-2}(v) \to v, u \to p_{k-c-2}(u) \to p_{c+1}(v) \to v)$.

It is straightforward to see that the space is $O(m + n^{1+1/k})$ and that the query takes $O(\mu |C(u,A_c)|) = O(\mu n^{c/k})$ time. The main technical contribution is in proving that $\hat{d}(u,v) \leq (2k-4c-1) \cdot d(u,v)$. To prove it, we use the borderline vertices technique. Since that $\tau_u \in C^*(u,A_c)$, when the algorithm iterates over $C^*(u,A_c)$, there is an iteration in which $u' = \tau_u$. In this iteration we guarantee that $\hat{d}(u,v) \leq d(u,\tau_u) + \texttt{MTZQuery}(\tau_u, v)$. Since τ_u is a borderline vertex, we either have that $\hat{d}(u,v) \leq u \to \tau_u \to p_c(\tau_u) \to v \leq 3d(u,v)$, and the

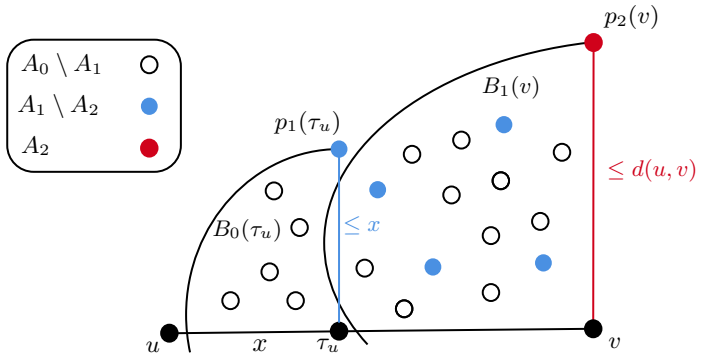


Figure 1 $p_1(\tau_u) \notin B_1(v)$.

estimation guarantee holds, or that $h_{c+1}(v) \leq d(u,v)$. Similarly, since $\tau_v \in C^*(v, A_c)$ and we iterate over $C^*(u, A_c)$, there is an iteration in which $v' = \tau_v$. In this iteration, either a short path is found ($v \to \tau_v \to p_c(\tau_u) \to u$), or we have that $h_{c+1}(u) \leq d(u,v)$.

From the properties of `MTZQuery`, for every $c+1 < i < k$, either a short path is found or $h_i(u) \leq d(u,v) + h_{i-1}(v)$. Since $h_{c+1}(u) \leq d(u,v)$ and $h_{c+1}(v) \leq d(u,v)$, it follows that

$$h_{k-c-2}(u) \leq \max(h_{c+1}(u), h_{c+1}(v)) + (k-2c-3)d(u,v) \leq (k-2c-2)d(u,v).$$

In the query procedure, we have that $\hat{d}(u,v) \leq u \to p_{k-c-2}(u) \to p_{c+1}(v) \to v \leq 2h_{k-c-2}(u) + d(u,v) + 2h_{c+1}(v)$. Thus:

$$\hat{d}(u,v) \leq 2(k-2c-2)d(u,v) + d(u,v) + 2d(u,v) = (2k-4c-1)d(u,v),$$

as required.

3.3 The middle vertex technique

For unweighted graphs, we develop the middle vertex technique that allows us to bound either $h_2(u)$ or $h_2(v)$ by $d(u,v)/2 + 1_{\text{ODD}}$, improving the previous $3d(u,v)/2 + 1_{\text{ODD}}$ bound from [22]. Roughly speaking, using the middle vertex technique technique we show that if $P(u,v) \not\subseteq C(u,A_1) \cap C(v,A_1)$ then $\min(h_2(u), h_2(v)) \leq d(u,v) + 1_{\text{ODD}}$. The middle vertex technique requires $O(n^{1/k})$ query time, matching the query time of the unweighted borderline technique. However, by using the middle vertex technique we reduce the additive error from 2 to 1_{ODD} in the bound of $\min(h_2(u), h_2(v))$.

For the sake of simplicity, let A_0, A_1, A_2, A_3 be a vertex hierarchy, where $A_0 = V$, $A_3 = \emptyset$ and let $d(u,v) = \delta$ be even. Let τ be the middle vertex on $P(u,v)$. That is, $d(u,\tau) = d(\tau,v) = \delta/2$ ($\delta/2$ is an integer since δ is even). The middle vertex τ has the following useful property. If $h_1(\tau) > \delta/2$ then $u, v \in B_0(\tau)$. The problem is that τ is unknown. However, the case that $u, v \in B_0(\tau)$ is equivalent to the case that $\tau \in C(u) \cap C(v)$. We use the $O(n^{1/k})$ query time to compute for every $w \in C(u) \cap C(v)$ the value of $d(u,w) + d(w,v)$. Therefore, even without knowing τ, in this case we have that $\hat{d}(u,v) = \delta$. (See Figure 2(a).)

If $h_1(\tau) \leq \delta/2$ then $d(u, p_1(\tau)) \leq d(u,\tau) + h_1(\tau) \leq \delta$ and $d(v, p_1(\tau)) \leq \delta$. If $p_1(\tau) \in B_1(u) \cap B_1(v)$, we exploit, again, the $O(n^{1/k})$ query time to compute for every $w \in B_1(u) \cap B_1(v)$ the value $d(u,w) + d(w,v)$. Therefore, even without knowing $p_1(\tau)$ the query algorithm returns $\hat{d}(u,v) \leq d(u, p_1(\tau)) + d(v, p_1(\tau)) \leq 2\delta$. (See Figure 2(b).) If $p_1(\tau) \notin B_1(u) \cap B_1(v)$ then without loss of generality $p_1(\tau) \notin B_1(u)$ and therefore $h_2(u) \leq d(u, p_1(\tau)) \leq \delta$. (See Figure 2(c).)

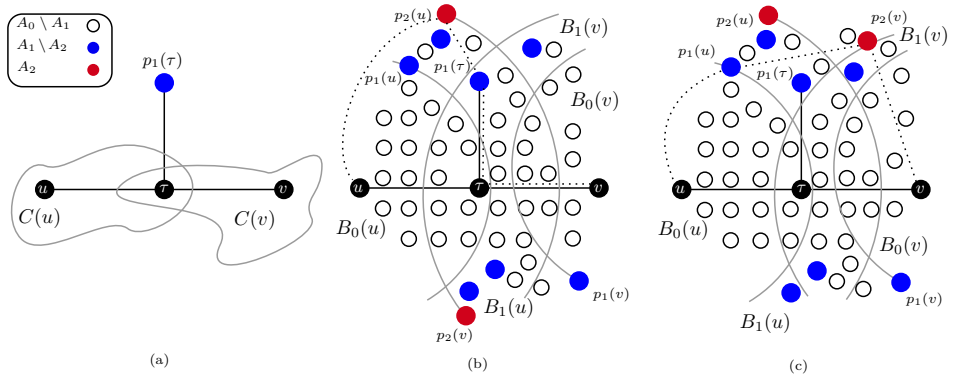


Figure 2 (a) $\tau \in C(u) \cap C(v)$. (b) $p_1(\tau) \in B_1(u) \cap B_1(v)$. (c) $p_1(\tau) \notin B_1(u) \cap B_1(v)$. In the figure $p_1(\tau) \notin B_1(v)$.

Our distance oracles for unweighted graphs use the middle vertex technique. When combined with the unweighted borderline vertices technique, this enables the construction of two additional oracles.

3.4 Distance oracles with stretch < 2

Let $A \subseteq V$ be a set of size $O(n^{1-c})$, let $u, v \in V$, let $\delta = d(u, v)$ and let $P = P(u, v)$. Let $t \geq 0$ be an integer. Let $S_t(u) = \bigcup_{w \in S_{t-1}(u)} B(w)$, where $S_0(u) = \{u\}$. Let d_t be a distance function induced by S_t (for the exact definition of d_t. Let $u = u_0$ (resp. $v = v_0$). For every $1 \leq i \leq t$, let $u_i \in B(u_{i-1})$ (resp. $v_i \in B(v_{i-1})$) be the farthest vertex in P from u (resp. v). (See Figure 3 for an illustration.) Let $P_t(u) = P(u, u_t)$. By the definition of $S_t(u)$ we have that $P_t(u) \subseteq S_t(u)$. We prove the following two properties:

- If $P_t(u) \cap P_t(v) \neq \emptyset$ then $d_t(u, w) + d_t(v, w) = \delta$ for some $w \in P_t(u) \cap P_t(v)$. (See Figure 3(b).)
- If $P_t(u) \cap P_t(v) = \emptyset$ then $\sum_{i=0}^{t-1} h(u_i) + h(v_i) \leq \delta + 2t - 1$. (See Figure 3(a).)

Next, we overview the distance oracle that uses these properties. The distance oracles stores the graph G and $d(x, y)$, for every $\langle x, y \rangle \in A \times V$, at the cost of $O(|V| \cdot |A|) = O(m + n^{2-c})$ space. In the query of the distance oracle, we compute $S_t(u)$ and $S_t(v)$. To address the case that $P_t(u) \cap P_t(v) \neq \emptyset$ we compute for every $w \in S_t(u) \cap S_t(v)$ in $O(\min(|S_t(u)|, |S_t(v)|))$ time the value $d_t(u, w) + d_t(v, w)$. In such a case, from the first property we have $d_t(u, w) + d_t(v, w) = \delta$, for some w.

To address the case that $P_t(u) \cap P_t(v) = \emptyset$ we compute $d(u, p(w)) + d(p(w), v)$ for every $w \in S_t(u) \cup S_t(v)$. Let $w \in \bigcup_{i=0}^{t-1} \{u_i, v_i\}$ be the vertex with minimal $h(w)$ value. Using the bound $\sum_{i=0}^{t-1} h(u_i) + h(v_i) \leq d(u, v) + 2t - 1$, we can show that $h(w) \leq \lfloor (d(u, v) + 2t - 1)/2t \rfloor = \lceil d(u, v)/2t \rceil$ and get that $d(u, p(w)) + d(p(w), v) \leq \delta + 2\lceil \delta/2t \rceil$.

Next, we reduce the space from $O(n^{2-c})$ to $O(n^{2-2c})$ by saving $d(x, y)$, for every $\langle x, y \rangle \in A \times A$ instead of saving $d(x, y)$, for every $\langle x, y \rangle \in A \times V$.

This information prevents us from using paths of the form $u \rightsquigarrow p(w) \rightsquigarrow v$. We can still use paths of the form $u \rightsquigarrow u_i \rightsquigarrow p(u_i) \rightsquigarrow p(v_j) \rightsquigarrow v_j \rightsquigarrow v$. However, since we do not know the vertices u_i and v_j we need to consider all vertex pairs in $S_{t-1}(u) \times S_{t-1}(v)$. This increases the query time from $O(n^{tc})$ to $O(n^{2(t-1)c})$. To avoid $O(n^{2(t-1)c})$ query time, we show that it suffices to consider only vertex pairs in $S_i(u) \times S_{t-1-i}(v)$, where $0 \leq i \leq t - 1$ in $O(tn^{(t-1)c})$-time, and still get the same approximation without iterating over all vertex pairs in $S_{t-1}(u) \times S_{t-1}(v)$.

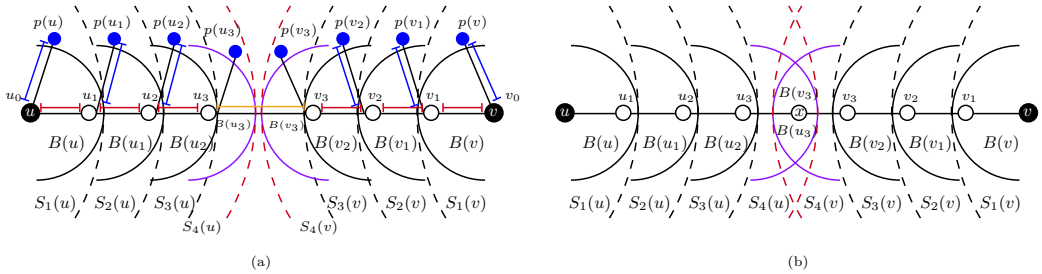


Figure 3 (a) $P_4(u) \cap P_4(v) = \emptyset$. The blue interval between u_i (v_i) and $p(u_i)$ ($p(v_i)$), where $i \in \{0, 1, 2\}$, is bounded by the red interval between u_i (v_i) and u_{i+1} (v_{i+1}) plus 1. The orange interval plus 1 bounds $h(u_3) + h(v_3)$. (b) $P_4(u) \cap P_4(v) = \{x\}$.

We define the set $Q = \{\langle u_i, v_{t-i-1}\rangle \mid 0 \leq i \leq t-1\}$. Let $q = \langle q_u, q_v\rangle \in Q$ be the vertex pair with minimal $h(q_u) + h(q_v)$ value. We prove that $h(q_u) + h(q_v) \leq \lfloor (d(u,v) + 2t - 1)/t \rfloor = \lceil d(u,v)/t \rceil + 1$. While iterating over the pairs of $S_i(u) \times S_{t-1-i}(v)$, we encounter q, and get that $\hat{d}(u,v) \leq \delta + 2\lceil \delta/t \rceil + 2$. Thus, with $O(n^{2-2c})$ space and $O(n^{tc})$ query time we get an estimation $\hat{d}(u,v) \leq \delta + 2\lceil \delta/t \rceil + 2$. By setting $c = 1/3 - \varepsilon$ and $t = 2, 3$ we get two new distance oracles, one with $O(n^{4/3+2\varepsilon})$ space and $O(n^{2/3-2\varepsilon})$ query, and estimation $\hat{d}(u,v) \leq \delta + 2\lceil \delta/2 \rceil + 2 \leq 2\delta + 4$, and another with $O(n^{4/3+2\varepsilon})$ space and $O(n^{1-3\varepsilon})$ query, and estimation $\hat{d}(u,v) \leq \delta + 2\lceil \delta/3 \rceil + 2 \leq \frac{5}{3}\delta + 4$.

We also consider weighted graphs with non-negative real edge weights. Agarwal, Godfrey and Har-Peled [7] defined $B^*(u)$ to be $B(u) \cup N(B(u))$. We revise the definition of S_t to be $\bigcup_{w \in S_{t-1}(u)} B^*(u)$. This new definition allows us to extend the previous results to weighted graphs. The distance oracles obtained using this approach are presented in the full version[27].

We remark that the usage of the sets $S_t(u)$ and $S_t(v)$ in the query algorithm is similar to the usage of the graph H in the work of [14]. However, in our analysis of the unweighted case, we achieve a slightly better distance approximation. In addition, we introduce a new approach to obtain a distance estimation using two vertices from A that are close to $P(u,v)$ rather than a single vertex from A, as in [14].

References

1 Amir Abboud and Greg Bodwin. The 4/3 additive spanner exponent is tight. *J. ACM*, 64(4):28:1–28:20, 2017. doi:10.1145/3088511.

2 Amir Abboud, Greg Bodwin, and Seth Pettie. A hierarchy of lower bounds for sublinear additive spanners. *SIAM J. Comput.*, 47(6):2203–2236, 2018. doi:10.1137/16M1105815.

3 Amir Abboud, Karl Bringmann, and Nick Fischer. Stronger 3-sum lower bounds for approximate distance oracles via additive combinatorics. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 391–404. ACM, 2023. doi:10.1145/3564246.3585240.

4 Amir Abboud, Karl Bringmann, Seri Khoury, and Or Zamir. Hardness of approximation in p via short cycle removal: cycle detection, distance oracles, and beyond. 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 1487–1500. ACM, 2022. doi:10.1145/3519935.3520066.

5 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.

6 Rachit Agarwal. The space-stretch-time tradeoff in distance oracles. In Andreas S. Schulz and Dorothea Wagner, editors, *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, volume 8737 of *Lecture Notes in Computer Science*, pages 49–60. Springer, 2014. doi:10.1007/978-3-662-44777-2_5.

7 Rachit Agarwal, Brighten Godfrey, and Sariel Har-Peled. Faster approximate distance queries and compact routing in sparse graphs. *CoRR*, abs/1201.2703, 2012. arXiv:1201.2703.

8 Rachit Agarwal and Philip Brighten Godfrey. Distance oracles for stretch less than 2. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 526–538. SIAM, 2013. doi:10.1137/1.9781611973105.38.

9 Rachit Agarwal, Philip Brighten Godfrey, and Sariel Har-Peled. Approximate distance queries and compact routing in sparse graphs. In *INFOCOM 2011. 30th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 10-15 April 2011, Shanghai, China*, pages 1754–1762. IEEE, 2011. doi:10.1109/INFCOM.2011.5934973.

10 Surender Baswana and Telikepalli Kavitha. Faster algorithms for all-pairs approximate shortest paths in undirected graphs. *SIAM J. Comput.*, 39(7):2865–2896, 2010. doi:10.1137/080737174.

11 Surender Baswana, Telikepalli Kavitha, Kurt Mehlhorn, and Seth Pettie. Additive spanners and (alpha, beta)-spanners. *ACM Trans. Algorithms*, 7(1):5:1–5:26, 2010. doi:10.1145/1868237.1868242.

12 Davide Bilò, Shiri Chechik, Keerti Choudhary, Sarel Cohen, Tobias Friedrich, Simon Krogmann, and Martin Schirneck. Approximate distance sensitivity oracles in subquadratic space. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 1396–1409. ACM, 2023. doi:10.1145/3564246.3585251.

13 Davide Bilò, Shiri Chechik, Keerti Choudhary, Sarel Cohen, Tobias Friedrich, and Martin Schirneck. Improved approximate distance oracles: Bypassing the thorup-zwick bound in dense graphs. *CoRR*, abs/2307.11677, 2023. doi:10.48550/arXiv.2307.11677.

14 Davide Bilò, Shiri Chechik, Keerti Choudhary, Sarel Cohen, Tobias Friedrich, and Martin Schirneck. Improved approximate distance oracles: Bypassing the thorup-zwick bound in dense graphs. *arXiv preprint arXiv:2307.11677*, 2023. doi:10.48550/arXiv.2307.11677.

15 Davide Bilò, Shiri Chechik, Keerti Choudhary, Sarel Cohen, Tobias Friedrich, and Martin Schirneck. Improved distance (sensitivity) oracles with subquadratic space. In *2024 IEEE 65th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1550–1558. IEEE, 2024. doi:10.1109/FOCS61266.2024.00097.

16 Greg Bodwin and Virginia Vassilevska Williams. Better distance preservers and additive spanners. *ACM Trans. Algorithms*, 17(4), October 2021. doi:10.1145/3490147.

17 Shiri Chechik. New additive spanners. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 498–512. SIAM, 2013. doi:10.1137/1.9781611973105.36.

18 Shiri Chechik. Approximate distance oracles with constant query time. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 654–663. ACM, 2014. doi:10.1145/2591796.2591801.

19 Shiri Chechik. Approximate distance oracles with improved bounds. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 1–10. ACM, 2015. doi:10.1145/2746539.2746562.

20 Shiri Chechik, Itay Hoch, and Gur Lifshitz. New approximation algorithms and reductions for n-pairs shortest paths and all-nodes shortest cycles. In Yossi Azar and Debmalya Panigrahi, editors, *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*, pages 5207–5238. SIAM, 2025. doi:10.1137/1.9781611978322.177.

21 Shiri Chechik and Tianyi Zhang. Path-reporting distance oracles with logarithmic stretch and linear size. In Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson, editors, *51st International Colloquium on Automata, Languages, and Programming, ICALP 2024, July 8-12, 2024, Tallinn, Estonia*, volume 297 of *LIPIcs*, pages 42:1–42:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ICALP.2024.42.

22 Mina Dalirrooyfard, Ce Jin, Virginia Vassilevska Williams, and Nicole Wein. Approximation algorithms and hardness for n-pairs shortest paths and all-nodes shortest cycles. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 290–300. IEEE, 2022. doi:10.1109/FOCS54457.2022.00034.

23 Michael Elkin, Ofer Neiman, and Christian Wulff-Nilsen. Space-efficient path-reporting approximate distance oracles. *Theor. Comput. Sci.*, 651:1–10, 2016. doi:10.1016/J.TCS.2016.07.038.

24 Michael Elkin and David Peleg. (1+epsilon, beta)-spanner constructions for general graphs. *SIAM J. Comput.*, 33(3):608–631, 2004. doi:10.1137/S0097539701393384.

25 Michael Elkin and Seth Pettie. A linear-size logarithmic stretch path-reporting distance oracle for general graphs. *ACM Trans. Algorithms*, 12(4):50:1–50:31, 2016. doi:10.1145/2888397.

26 Michael Elkin and Idan Shabat. Path-reporting distance oracles with logarithmic stretch and size o(n log log n). In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 2278–2311. IEEE, 2023. doi:10.1109/FOCS57990.2023.00141.

27 Avi Kadria and Liam Roditty. New approximate distance oracles and their applications. *arXiv preprint arXiv:2509.00890*, 2025.

28 Tsvi Kopelowitz, Ariel Korin, and Liam Roditty. On the space usage of approximate distance oracles with sub-2 stretch. In Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson, editors, *51st International Colloquium on Automata, Languages, and Programming, ICALP 2024, July 8-12, 2024, Tallinn, Estonia*, volume 297 of *LIPIcs*, pages 101:1–101:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ICALP.2024.101.

29 Manor Mendel and Assaf Naor. Ramsey partitions and proximity data structures. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 21-24 October 2006, Berkeley, California, USA, Proceedings*, pages 109–118. IEEE Computer Society, 2006. doi:10.1109/FOCS.2006.65.

30 Merav Parter. Bypassing erdős' girth conjecture: Hybrid stretch and sourcewise spanners. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, volume 8573 of *Lecture Notes in Computer Science*, pages 608–619. Springer, 2014. doi:10.1007/978-3-662-43951-7_49.

31 Ely Porat and Liam Roditty. Preprocess, set, query! *Algorithmica*, 67(4):516–528, 2013. doi:10.1007/S00453-013-9825-9.

32 Mihai Pătraşcu and Liam Roditty. Distance oracles beyond the thorup-zwick bound. *SIAM J. Comput.*, 43(1):300–311, 2014. doi:10.1137/11084128X.

33 Mihai Pătraşcu, Liam Roditty, and Mikkel Thorup. A new infinity of distance oracles for sparse graphs. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 738–747. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.44.

34 Liam Roditty, Mikkel Thorup, and Uri Zwick. Deterministic constructions of approximate distance oracles and spanners. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005, Lisbon, Portugal, July 11-15, 2005, Proceedings*, volume 3580 of *Lecture Notes in Computer Science*, pages 261–272. Springer, 2005. doi:10.1007/11523468_22.

35 Mikkel Thorup and Uri Zwick. Compact routing schemes. In Arnold L. Rosenberg, editor, *Proceedings of the Thirteenth Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA 2001, Heraklion, Crete Island, Greece, July 4-6, 2001*, pages 1–10. ACM, 2001. doi:10.1145/378580.378581.

36 Mikkel Thorup and Uri Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, 2005. doi:10.1145/1044731.1044732.

37 Mikkel Thorup and Uri Zwick. Spanners and emulators with sublinear distance errors. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pages 802–809. ACM Press, 2006. URL: http://dl.acm.org/citation.cfm?id=1109557.1109645.

38 Christian Wulff-Nilsen. Approximate distance oracles with improved preprocessing time. 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 202–208. SIAM, 2012. doi:10.1137/1.9781611973099.18.

A Tables of our distance oracles

A.1 Weighted graphs with stretch ≥ 2

Table 2 Distance oracles in weighted graphs.

Construction	Space	Time	Stretch	Ref.	Comments
$\tilde{O}(mn^{1/k})$	$\tilde{O}(n^{1+1/k})$	$\tilde{O}(1)$	$2k-1$	[36]	
$\tilde{O}(mn^{3/k})$	$\tilde{O}(m+n^{1+1/k})$	$\tilde{O}(\mu n^{c/k})$	$2k-1-4c$	Th1	$1 \leq c < k/2$
$\tilde{O}(mn^{3/k})$	$\tilde{O}(m+n^{1+1/k})$	$\tilde{O}(\mu n^{r})$	$2k(1-2r)-1$	Th1	$c/k = r < 1/2$
$\tilde{O}(mn^{1/k})$	$\tilde{O}(m+n^{1+1/k})$	$\tilde{O}(\mu n^{1/k})$	$2k-2$	[22]	$k \geq 3$
$\tilde{O}(mn^{1/k})$	$\tilde{O}(m+n^{1+1/k})$	$\tilde{O}(\mu n^{1/k})$	$2k-3$	Th15[27]	$k \geq 3$
$\tilde{O}(mn^{2/k})$	$\tilde{O}(m+n^{1+1/k})$	$\tilde{O}(\mu n^{1/k})$	$2k-4$	Th2	$k \geq 4$
$\tilde{O}(mn^{3/k})$	$\tilde{O}(m+n^{1+1/k})$	$\tilde{O}(\mu n^{1/k})$	$2k-5$	Th1	$k \geq 5$
$\tilde{O}(mn^{2/3})$	$\tilde{O}(m+n^{4/3})$	$\tilde{O}(\mu n^{1/3})$	3	[7]	
$\tilde{O}(mn^{1/3})$	$\tilde{O}(m+n^{4/3})$	$\tilde{O}(\mu n^{1/3})$	3	Th15[27]	
$\tilde{O}(mn^{1/2})$	$\tilde{O}(m+n^{5/4})$	$\tilde{O}(\mu n^{1/4})$	4	Th2	
$\tilde{O}(mn^{3/5})$	$\tilde{O}(m+n^{6/5})$	$\tilde{O}(\mu n^{1/5})$	5	Th1	
$\tilde{O}(mn^{1-c})$	$\tilde{O}(m+n^{2-c})$	$\tilde{O}(\mu^t n^{tc})$	$1+1/t$	[6]	$c < 1/2$
$\tilde{O}(mn^{1-c})$	$\tilde{O}(m+n^{2-c})$	$\tilde{O}(\mu^t n^{(t+1)c})$	$1+1/(t+0.5)$	[6]	$c < 1/2$
$\tilde{O}(mn^{1-c})$	$\tilde{O}(m+n^{2-2c})$	$\tilde{O}(\mu^t n^{tc})$	$1+2/t$	Th8	$c < 1/3$

A.2 n-PSP and $ANSC$ results

Table 3 $nPSP$ in weighted graphs unless stated otherwise. $\delta = d(u, v)$.

Time	Approximation	Ref.	Comment
$\tilde{O}(m + n^{1+2/k})$	$(2k-1)(2k-2)\delta + 4k - 2$	[22]	Unweighted, $k \geq 2$
$\tilde{O}(m + n^{1+2/k})$	$(2k-1)(2k-3)\delta$	Theorem 12	$k \geq 2$
$\Omega(m^{2-\frac{2}{k+1}} n^{\frac{1}{k+1}-o(1)})$	$(1+1/k)\delta$	[22]	
$\tilde{O}(m^{2-\frac{2}{k+1}} n^{\frac{1}{k+1}})$	$(1+1/k)\delta$	[6][8]	
$\tilde{O}(m^{1-\frac{1}{k+1}} n)$	$\delta + 2\lceil \delta/2k \rceil$	Theorem 11	Unweighted, $k \geq 2$

Table 4 $ANSC$ algorithm improvements in undirected graphs. $SC(u)$ is the shortest cycle for u.

Time	Approximation	Ref.	Comment
$\tilde{O}(m^{2-2/k} n^{1/k})$	$SC(u) + 2\lceil SC(u)/2(k-1) \rceil$	[22]	Unweighted, $k \geq 2$
$\tilde{O}(mn^{1-1/k})$	$SC(u) + 2\lceil SC(u)/2(k-1) \rceil$	Theorem 13	Unweighted, $k \geq 2$
$\tilde{O}(m^{2-\frac{1}{k}})$	$(1 + \frac{1}{k-1})SC(u)$	Theorem 14	Weighted, $k \geq 2$

[8] This result is obtained by constructing the distance oracle of Agarwal [6] and querying it n times.

Approximation Schemes for k-Subset Sum Ratio and k-Way Number Partitioning Ratio

Sotiris Kanellopoulos[1]
National Technical University of Athens and Archimedes, Athena Research Center, Greece

Giorgos Mitropoulos[1]
National Technical University of Athens and Archimedes, Athena Research Center, Greece

Antonis Antonopoulos
National Technical University of Athens, Greece

Nikos Leonardos
National Technical University of Athens, Greece

Aris Pagourtzis
National Technical University of Athens and Archimedes, Athena Research Center, Greece

Christos Pergaminelis
National Technical University of Athens and Archimedes, Athena Research Center, Greece

Stavros Petsalakis
National Technical University of Athens, Greece

Kanellos Tsitouras
National Technical University of Athens, Greece

Abstract

The Subset Sum Ratio problem (SSR) asks, given a multiset A of positive integers, to find two disjoint subsets of A such that the *largest-to-smallest* ratio of their sums is minimized. In this paper we study the k-version of SSR, namely k-Subset Sum Ratio (k-SSR), which asks to minimize the *largest-to-smallest* ratio of sums of k disjoint subsets of A. We develop an approximation scheme for k-SSR running in $O(n^{2k}/\varepsilon^{k-1})$ time, where $n = |A|$ and ε is the error parameter. To the best of our knowledge, this is the first FPTAS for k-SSR for fixed $k > 2$.

We also study the k-way Number Partitioning Ratio (k-PART) problem, which differs from k-SSR in that the k subsets must constitute a partition of A; this problem in fact corresponds to the objective of minimizing the largest-to-smallest sum ratio in the family of Multiway Number Partitioning problems. We present a more involved FPTAS for k-PART, also achieving $O(n^{2k}/\varepsilon^{k-1})$ time complexity. Notably, k-PART is also equivalent to the Minimum Envy-Ratio problem with identical valuation functions, which has been studied in the context of fair division of indivisible goods. Thus, for the case of identical valuations, our FPTAS represents a significant improvement over the $O(n^{4k^2+1}/\varepsilon^{2k^2})$ bound obtained by Nguyen and Rothe's FPTAS [36] for Minimum Envy-Ratio with general additive valuations.

Lastly, we propose a second FPTAS for k-SSR, which employs carefully designed calls to the first one; the new scheme has a time complexity of $\widetilde{O}(n/\varepsilon^{3k-1})$, thus being much faster when $n \gg 1/\varepsilon$.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases Fully polynomial-time approximation schemes, Subset Sum Ratio, Number Partitioning, Fair division, Envy minimization, Pseudo-polynomial time algorithms

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.44

Related Version *Full Version*: https://arxiv.org/abs/2503.18241 [24]

[1] Equal Contribution

© Sotiris Kanellopoulos, Giorgos Mitropoulos, Antonis Antonopoulos, Nikos Leonardos, Aris Pagourtzis, Christos Pergaminelis, Stavros Petsalakis, and Kanellos Tsitouras; licensed under Creative Commons License CC-BY 4.0

36th International Symposium on Algorithms and Computation (ISAAC 2025).
Editors: Ho-Lin Chen, Wing-Kai Hon, and Meng-Tsung Tsai; Article No. 44; pp. 44:1–44:22

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

Funding Supported by project MIS 5154714 of the National Recovery and Resilience Plan Greece 2.0 funded by the European Union under the NextGenerationEU Program.

1 Introduction

The Subset Sum and Partition problems are fundamental in combinatorial optimization, with numerous applications in fair resource allocation, cryptography and scheduling. The Equal Subset Sum (ESS) problem [43] asks for two disjoint subsets of a given set with equal sums, while its optimization counterpart, Subset Sum Ratio (SSR) [43, 4], asks to minimize the ratio of sums of two disjoint subsets. These NP-hard problems arise in bioinformatics [15, 14], game theory and economics [30, 18], and cryptography [44], among others. A special case of ESS belongs to PPP [37], which is a subclass of TFNP, further adding to the ongoing interest for these problems.

In this paper we consider variations involving multiple subsets, namely k-Subset Sum Ratio (k-SSR) and k-way Number Partitioning Ratio (k-PART)[2]. In k-SSR, the goal is to find k disjoint subsets such that the *largest-to-smallest* ratio of their sums is minimized, while k-PART further requires these subsets to form a partition of the input. Since these problems are NP-hard, we consider approximation algorithms. In particular, we present fully polynomial-time approximation schemes (FPTASs) for both problems.

Our results apply, among others, to scheduling tasks across k processors (e.g. [39, 17, 22, 16]), as well as to *fair division of indivisible goods*, in particular the study of *envy* in discrete settings (e.g. [30, 36, 28] and more recently [2, 41, 45]). Specifically, k-PART is equivalent to the Minimum Envy-Ratio problem [30] with identical valuation functions, for which we achieve a significant improvement over (more general) earlier work [36].

Related work

Since Bellman's seminal work on Subset Sum [5], no major improvement had occurred until a recent spike in interest culminated in various faster pseudo-polynomial algorithms [26, 7, 23, 12] and FPTASs [25, 20].

The ESS and SSR problems have seen recent advances, including FPTASs for SSR [35, 31, 1, 8] and an improved exact algorithm for ESS [33]. Notably, Bringmann [8] provided an SSR FPTAS running in time $O(n/\varepsilon^{0.9386})$, which is faster than known Subset Sum lower bounds. He achieved this in part by considering two cases based on input density. If the input is dense, a pigeonhole argument guarantees two subsets with sums within $1+\varepsilon$ and removing their intersection yields an approximate solution. If the input is sparse, keeping only the polylog$(1/\varepsilon)$ largest elements is proven to be sufficient for the approximation. Regarding variations of (Equal) Subset Sum involving $k \geq 2$ subsets, [3] extends the techniques of [26, 7] to provide both deterministic and randomized pseudo-polynomial algorithms.

For Partition, advances include a number of (very) recent FPTASs [34, 9, 11], with the latest work by Chen et al. [13] achieving an FPTAS running in time $\widetilde{O}(n+1/\varepsilon)$, which is near optimal under the Strong Exponential Time Hypothesis. In Multiway Number Partitioning [6, 40, 27, 38], different objective functions define various approaches to distributing elements among subsets, such as minimizing the maximum sum, maximizing the minimum sum, minimizing the *largest-to-smallest* difference of sums, or minimizing the *largest-to-smallest* ratio of sums. These objectives are not comparable: there are instances showing that the corresponding optimal solutions differ; we discuss this in detail in Subsection 2.1, along with applications of each variation.

[2] This problem is usually referred to as "multiway", but we call it "k-way" to emphasize that k is fixed.

Bazgan et al. [4] provided the first FPTAS for SSR and demonstrated that Subset Sum Difference does not admit an FPTAS, unless P = NP. Similar arguments that apply to Partition have been further studied in [42]. Additionally, various related problems such as 3-Partition and Bin Packing are strongly NP-hard [19], implying that they do not admit an FPTAS or a pseudo-polynomial time algorithm unless P = NP. However, these problems involve a non-constant number of subsets. For our problems, we consider a *fixed* number k of disjoint subsets. This restriction is quite natural in applications, as it is more likely to distribute a great amount of items to a small number of agents.

Sahni presents an FPTAS for Multiway Number Partitioning in [39] that minimizes the largest sum. However, this technique does not seem applicable to minimize the ratio. In [30], Lipton et al. show a PTAS for Minimum Envy-Ratio with identical valuations and non-constant number k of agents, while claiming the existence of an FPTAS when k is constant. Furthermore, in [36] Nguyen and Rothe show an FPTAS running in $O(n^{4k^2+1}/\varepsilon^{2k^2})$ time for the same problem with fixed k and general additive valuation functions.

Our contribution

In this paper, we present two FPTASs for k-SSR achieving running times[3] of $O(n^{2k}/\varepsilon^{k-1})$ and $\widetilde{O}(n/\varepsilon^{3k-1})$, and an FPTAS for k-PART running in time $O(n^{2k}/\varepsilon^{k-1})$. The latter represents a considerable improvement when compared to the FPTAS of [36], which is the best known bound for k-PART, although it concerns a more general setting. Note that the largest-to-smallest ratio of sums is the only objective function considered in the literature that both admits an FPTAS and is of interest to the study of *envy*. Our approach builds upon methods established for SSR [35, 31, 8], while introducing novel strategies to address the multi-subset setting and to comply with the partitioning constraint.

We present our first FPTAS for k-SSR in Section 3; to the best of our knowledge, this is the first ever FPTAS for this problem. To this end, we define a restricted version of k-SSR, for which we present a pseudo-polynomial time algorithm. Our approach relies on proving that the optimal solution of the restricted problem satisfies certain properties, while carefully handling edge cases involving large singleton sets.

Concerning k-PART, the partitioning constraint renders the application of our techniques more involved; we overcome this in Section 4 by identifying a *perfect* restriction parameter that ensures the existence of a *well-behaved* optimal solution. Interestingly, although this does not yield a pseudo-polynomial time algorithm for the respective restricted case, combining the aforementioned property with approximation techniques yields an FPTAS for k-PART.

Additionally, building on Bringmann's work [8], we find that applying density arguments to k-SSR encounters an obstacle in the dense case: obtaining k subsets with approximately equal sums is unhelpful, since removing their intersection does not guarantee a feasible k-SSR solution. We resolve this in Section 5 by restricting our density argument to singletons and subsequently using our FPTAS from Section 3 as a subroutine on reduced instances.

2 Preliminaries

2.1 Objective functions for Multiway Number Partitioning

There are four objective functions commonly used for Multiway Number Partitioning, each one having applications in different fields.

[3] $\widetilde{O}$ hides polylog$(1/\varepsilon)$ factors.

1. Minimizing the largest sum. This is also known as makespan minimization and is used for the Minimum Finish Time problem in the context of identical machine scheduling [39, 22]. It is also used in problems related to bin packing [29].
2. Maximizing the smallest sum. This objective function has been studied in works related to scheduling [17] and bin packing [29]. It has also been studied in works related to the *maximin share* [21, 28, 10], which is a criterion for fair item allocation. Note that algorithms for objective function (2) can be modified to work for (1) and vice versa [27].
3. Minimizing the difference between the largest and the smallest sum. This objective function is less common, but it has been studied in works related to Multiway Number Partitioning, such as [32, 27]. No FPTAS exists for this objective [4, 42].
4. Minimizing the ratio between the largest and the smallest sum (e.g. [3]). Although this objective function is often overlooked in works regarding Multiway Number Partitioning (e.g. [27, 40]), it has been studied in the field of fair division, in the form of the Minimum Envy-Ratio problem [30, 36], as well as in the context of scheduling [16].

In this paper, we consider (4) as an objective function for k-way Number Partitioning. Observe that all four objectives are equivalent when $k = 2$; this does not hold for $k \geq 3$. This is well known for the first three objectives [27], but, for the sake of completeness, we will provide some counterexamples that prove that (4) is not equivalent to any of the other three, for $k \geq 3$.

It is easy to find counterexamples for the first two. Consider input $\{1, 2, 3, 10\}$ for $k = 3$. The partition $(\{1\}, \{2, 3\}, \{10\})$ minimizes the largest sum, but does not have optimal ratio. Similarly, consider $\{5, 5, 5, 10\}$ for $k = 3$. The partition $(\{5\}, \{5\}, \{5, 10\})$ maximizes the smallest sum, but does not have optimal ratio.

Finally, consider input $\{16, 16, 18, 20, 24, 27, 29, 40\}$ for $k = 4$. The following table compares the solution that minimizes the difference with the solution that minimizes the ratio.

Table 1 Comparison of Multiway Number Partitioning solutions.

Solution	Sums	Difference	Ratio
$\{40\}, \{16, 16, 18\}, \{24, 27\}, \{29, 20\}$	{40,50,51,49}	**11**	1.275
$\{40, 16\}, \{24, 20\}, \{16, 29\}, \{18, 27\}$	{56,44,45,45}	12	**1.27273**

2.2 Notation and problem definitions

Let $[n] = \{1, \ldots, n\}$ for a positive integer n. We assume that the input $A = \{a_1, \ldots, a_n\}$ of our problems is sorted[4], i.e. $a_1 \leq \ldots \leq a_n$.

▶ **Definition 1** (Sum of a subset of indices). *Given a multiset $A = \{a_1, \ldots, a_n\}$ of positive integers and a set of indices $S \subseteq [n]$ we define:*

$$\Sigma(S, A) = \sum_{i \in S} a_i$$

[4] If the input is not sorted, an additional $O(n \log n)$ time would be required. This does not affect the time complexity of any of our algorithms, assuming that $\widetilde{O}$ also hides polylog(n) factors.

▶ **Definition 2** (Max and Min of k subsets). *Let $A = \{a_1, \ldots, a_n\}$ be a multiset of positive integers and $S_1, \ldots, S_k$ be k disjoint subsets of $[n]$. We define the maximum and minimum sums obtained by these sets on A as:*

$$M(S_1, \ldots, S_k, A) = \max_{i \in [k]} \Sigma(S_i, A) \quad \text{and} \quad m(S_1, \ldots, S_k, A) = \min_{i \in [k]} \Sigma(S_i, A)$$

▶ **Definition 3** (Largest-to-smallest ratio of k subsets). *Let $A = \{a_1, \ldots, a_n\}$ be a multiset of positive integers and $S_1, \ldots, S_k$ be k disjoint subsets of $[n]$. We define the* largest-to-smallest ratio *of these k subsets on A as:*

$$\mathcal{R}(S_1, \ldots, S_k, A) = \begin{cases} \frac{M(S_1,\ldots,S_k,A)}{m(S_1,\ldots,S_k,A)} & \text{if } m(S_1, \ldots, S_k, A) > 0 \\ +\infty & \text{if } m(S_1, \ldots, S_k, A) = 0 \end{cases}$$

Throughout the paper, we refer to the largest-to-smallest ratio of k subsets simply as *ratio*.

We define the k-Subset Sum Ratio (k-SSR) problem as follows.

▶ **Definition 4** (k-SSR). *Given a multiset $A = \{a_1, \ldots, a_n\}$ of positive integers, find k disjoint subsets $S_1, \ldots, S_k$ of $[n]$, such that $\mathcal{R}(S_1, \ldots, S_k, A)$ is minimized.*

▶ **Observation 5.** *Only the maximum and minimum sums affect the ratio function. If the remaining sets are altered without a sum becoming greater than the maximum or smaller than the minimum, the ratio remains unaffected. Additionally, if the minimum sum increases or the maximum sum decreases (while the other remains unchanged), the ratio decreases.*

Note that multisets are allowed as input, since they are not a trivial case for k-SSR (in contrast to regular SSR), unless there is a number with multiplicity k or more. Throughout the paper, when referring to a solution $S = (S_1, \ldots, S_k)$, we will use the simplified notations $\mathcal{R}(S, A), M(S, A), m(S, A)$ to denote ratio, maximum and minimum sums respectively.

We similarly define the k-way Number Partitioning Ratio (k-PART) problem.

▶ **Definition 6** (k-PART). *Given a multiset $A = \{a_1, \ldots, a_n\}$ of positive integers, find k disjoint subsets $S_1, \ldots, S_k$ of $[n]$ with $\bigcup_{i=1}^{k} S_i = [n]$, such that $\mathcal{R}(S_1, \ldots, S_k, A)$ is minimized.*

3 An FPTAS for k-Subset Sum Ratio

We define a restricted version of k-SSR, called k-SSR$_{\mathrm{R}}$, in which the largest element[5] of the first set (S_1) is forced to be the smallest among the maxima of all k sets and equal to a given number p. This definition generalizes the *Semi-Restricted Subset-Sums Ratio* of [31].

▶ **Definition 7** (k-SSR$_{\mathrm{R}}$). *Given a sorted multiset $A = \{a_1, \ldots, a_n\}$ of positive integers and an integer $p, 1 \leq p \leq n - k + 1$, find k disjoint subsets $S_1, \ldots, S_k$ of $[n]$ with $\max(S_1) = p$ and $\max(S_i) > p$ for $1 < i \leq k$, such that $\mathcal{R}(S_1, \ldots, S_k, A)$ is minimized.*[6]

Most of this section is dedicated to producing an FPTAS for the restricted version k-SSR$_{\mathrm{R}}$, which is subsequently used as a subroutine to obtain an FPTAS for k-SSR, by iterating over all possible values of p. Similar reductions to a version with restricted largest elements have been used in approximation schemes for Subset Sum Ratio [35, 31, 8]. In our paper, this technique is used to guarantee that each set S_i contains a sufficiently large element, which is critical to ensure that the algorithm is an FPTAS.

[5] Throughout the paper, the term *element* of a set refers to the index $1 \leq i \leq n$ instead of the value a_i, unless explicitly stated otherwise.

[6] The case $p > n - k + 1$ would result in the instance having no feasible solution.

3.1 Properties of optimal solutions to k-SSR$_{\mathrm{R}}$ instances

First, we want to guarantee that there is always an optimal solution to k-SSR$_{\mathrm{R}}$ that satisfies a few important properties. Recall that the input is sorted. Define $Q = \sum_{i=1}^{p} a_i$ and $q = \max\{i \mid a_i \leq Q\}$. Since $Q = \Sigma([p], A)$ and $S_1 \subseteq [p]$, the following is immediate by Def. 2.

▶ **Observation 8.** *For all feasible solutions $(S_1, \ldots, S_k)$ to a k-SSR$_{\mathrm{R}}$ instance (A, p)*

$$m(S_1, \ldots, S_k, A) \leq \Sigma(S_1, A) \leq Q.$$

Observations 5 and 8 are used in the proof of the following theorem to transform solutions without increasing their ratio.

▶ **Theorem 9.** *For any k-SSR$_{\mathrm{R}}$ instance (A, p) there exists an optimal solution whose sets satisfy the following:*

1. *For all sets S_i containing only elements j with $a_j \leq Q$ it holds that $\Sigma(S_i, A) < 2Q$.*
2. *Every set containing an element j with $a_j > Q$ is a singleton.*
3. *The union of singleton sets $\{j\}$ s.t. $a_j > Q$ is $\bigcup_{i=1}^{x}\{q+i\}$, where $x \geq 0$ is the number of these singleton sets.*

Proof. Let $S = (S_1, \ldots, S_k)$ be an arbitrary feasible solution. If it violates property 1, i.e. there exists a set S_i in S that only contains elements j with $a_j \leq Q$ and has sum $\Sigma(S_i, A) \geq 2Q$, we transform it as follows. For all $j \in S_i$, we have

$$m(S, A) \leq \Sigma(S_1, A) \leq Q \leq \Sigma(S_i \setminus \{j\}, A) < \Sigma(S_i, A) \leq M(S, A).$$

Thus, if we remove an element j from S_i, the ratio cannot increase. We remove the smallest element from S_i. As long as the solution still has a set S_i violating property 1, we can apply the same process repeatedly, until there are no more such sets. Since we are only removing elements, no new sets violating property 1 may appear during this process. Note that the largest element of every set remains intact, therefore the k-SSR$_{\mathrm{R}}$ restrictions $max(S_1) = p$ and $\max(S_i) > p$ for $1 < i \leq k$ are satisfied.

If the new solution S' violates property 2, i.e. it contains a set S_i with an element j s.t. $a_j > Q$ and $|S_i| > 1$, we apply the following transformation. It holds that

$$m(S', A) \leq \Sigma(S_1, A) \leq Q < \Sigma(\{j\}, A) < \Sigma(S_i, A) \leq M(S', A).$$

As such, replacing set S_i with set $\{j\}$ will result in equal or smaller ratio. We repeat this for every such set S_i, thus yielding a solution in which every such set is a singleton. Note that the derived solution is a feasible k-SSR$_{\mathrm{R}}$ solution and it still satisfies property 1, since we did not modify sets that contain only elements j with $a_j \leq Q$.

Finally, if the new solution S'' violates property 3, i.e. it contains a singleton set $\{u\}$ s.t. $a_u > Q$ and there is an unselected[7] element v such that $q < v < u$, we do the following. It holds that

$$m(S'', A) \leq \Sigma(S_1, A) \leq Q < \Sigma(\{v\}, A) \leq \Sigma(\{u\}, A) \leq M(S'', A).$$

Thus, selecting v instead of u does not increase the ratio. Repeating this for all u and v satisfying the above mentioned property forces the union of these singleton sets to contain the smallest indices possible, yielding a feasible solution that satisfies all three properties.

In conclusion, for any feasible solution we can find another one that satisfies all properties 1, 2, 3 and has equal or smaller ratio. Thus, applying this to an arbitrary optimal solution proves the theorem. ◀

[7] An element that is not in any set S_i of the solution.

3.2 A pseudo-polynomial time algorithm for k-SSR$_{\mathrm{R}}$

In this subsection we present an exact algorithm for k-SSR$_{\mathrm{R}}$, which runs in $O(nQ^{k-1})$ time and returns an optimal solution satisfying the properties of Theorem 9. We present the algorithm in two parts for simplicity. Algorithm 1 picks certain singleton sets with large elements and calls Algorithm 2 to obtain candidate partial solutions for the remaining sets. We next describe both algorithms in detail; we refer the reader to the full version of this paper [24] for the respective pseudocode.

Algorithm 1 `Exact_`k-SSR$_{\mathrm{R}}(A, p)$.

Input: A sorted multiset $A = \{a_1, \ldots, a_n\}, a_i \in \mathbb{Z}^+$ and an integer $p, 1 \leq p \leq n - k + 1$.
Output: Disjoint subsets $(S_1, \ldots, S_k)$ of $[n]$ with $\max(S_1) = p$, $\max(S_i) > p$ for $1 < i \leq k$, minimizing $\mathcal{R}(S_1, \ldots, S_k, A)$.

Algorithm 1 iterates over all possible values for the number x of singleton sets $\{j\}$, with $a_j > Q$, specifically $x \in \{0, \ldots, \min\{k-1, n-q\}\}$. To construct these x sets, it picks the smallest elements of A that are greater than Q, in accordance to property 3 of Theorem 9. For each x, Algorithm 1 then calls Algorithm 2 as a subroutine; the latter uses dynamic programming (DP) to output partial solutions for $A' = \{a_i \in A \mid a_i \leq Q\}$ and $k' = k - x$. Note that Algorithm 2 is only called in cases where $p \leq |A'| - k' + 1 = q - k + x + 1$; otherwise, there is no feasible k-SSR$_{\mathrm{R}}$ solution for that value of x (see Definition 7).

Algorithm 1 appends the x precalculated singletons to each partial solution returned by Algorithm 2 and compares the ratio of each solution in order to find the best one. The k-SSR$_{\mathrm{R}}$ solution returned by Algorithm 1 is optimal, as will be shown in Theorem 12.

Algorithm 2 `DP_`k-SSR$_{\mathrm{R}}(A, k, p)$.

Input: A sorted multiset $A = \{a_1, \ldots, a_q\}, a_i \in \mathbb{Z}^+$, an integer $k \geq 1$ and an integer $p, 1 \leq p \leq q - k + 1$, with the restriction $a_q \leq \sum_{i=1}^{p} a_i$.
Output: A set *solutions* containing tuples of k disjoint subsets $(S_1, \ldots, S_k)$ of $[q]$, with $\max(S_1) = p$, $\max(S_i) > p$ for $1 < i \leq k$.

Algorithm 2 draws from techniques of [31] and [35], while employing a more involved DP formulation to address the multi-subset setting. A DP table T is used to systematically construct partial solution sets, while ensuring that the constraints of k-SSR$_{\mathrm{R}}$ are satisfied.

Each cell of the DP table stores a tuple $(S_1, \ldots, S_k, sum_1)$, where S_j $(1 \leq j \leq k)$ are the sets associated with said cell and $sum_1 = \Sigma(S_1, A)$. The coordinates of a cell $T_i[D][V]$ consist of the following components:

1. An index i, indicating the number of elements examined so far.
2. A *difference vector* $D = [d_2, d_3, \ldots, d_k]$ (sorted in nondecreasing order) that encodes the differences between the sum of S_1 and the sums of the other sets, i.e. $d_j = \Sigma(S_1, A) - \Sigma(S_j, A)$.
3. A Boolean *validity vector* $V = [v_2, v_3, \ldots, v_k]$, where each v_j is `true` iff $\max(S_j) > p$.

The differences are used as coordinates in the DP table instead of the sums of the sets, in order to decrease the complexity of the algorithm by an order of Q (see Lemma 13). We also introduce V as a coordinate, which is crucial to ensure optimality without increasing the complexity of the algorithm; essentially, the ratio of two tuples can be compared only if all of their validity Booleans are equal (see Lemma 10).

We now describe the DP process in a bottom-up manner. All cells $T_i[D][V]$ are initialized to empty tuples, except the cell $T_0[a_p, \ldots, a_p][\texttt{false}, \ldots, \texttt{false}]$, which is initialized to $(\{p\}, \emptyset, \ldots, \emptyset, a_p)$, thus forcing p to be contained in S_1. Every element $i \in [q]$ is processed in increasing order. Assuming row T_{i-1} contains tuples constructed by using elements up to $i-1$ (along with p, which is always in S_1), the next row T_i is filled as follows. For each non-empty tuple $C = (S_1, \ldots, S_k, sum_1)$ contained in some cell $T_{i-1}[D][V]$, $D = [d_2, \ldots, d_k]$, $V = [v_2, \ldots, v_k]$, we sequentially consider the cases below:

- Element i is not added to any set of C. In this case, C is copied into $T_i[D][V]$.
- Element i is added to S_1 (only if $i < p$). This produces a new tuple $C' = (S_1 \cup \{i\}, \ldots, S_k, sum_1 + a_i)$ to be inserted into $T_i[D'][V]$, where $D' = [d'_2, \ldots, d'_k]$ is the appropriately updated difference vector, i.e. $d'_j = d_j + a_i$, $1 < j \leq k$.
- Element $i \neq p$ is added to each set S_j $(1 < j \leq k)$, one at a time. This produces a new tuple $(S_1, \ldots, S_j \cup \{i\}, \ldots, S_k, sum_1)$, a difference vector $[d'_2, \ldots, d'_k]$ and a validity vector $[v'_2, \ldots, v'_k]$. Specifically, v'_j is set to `true` if $i > p$ and d'_j is set to $d_j - a_i$, while $v'_u = v_u$ and $d'_u = d_u$ for all $u \neq j$. This difference vector is then sorted in nondecreasing order, while the sets and the validity vector are rearranged accordingly, thus producing a tuple C' and vectors D', V'. The tuple C' is then inserted into $T_i[D'][V']$.

Algorithm 2 prevents the new tuple from being inserted into a cell in the following cases:

1. The cell already contains another tuple with equal or larger sum_1 (see Lemma 10).
2. The updated vector D' contains a difference smaller than or equal to $-2Q$. Such a tuple can be ignored, due to property 1 of Theorem 9.

In the end, Algorithm 2 returns all non-empty tuples contained in cells $T_q[D][\texttt{true}, \ldots, \texttt{true}]$ (for all D), thus ensuring that every partial solution returned to Algorithm 1 complies with the k-SSR$_\mathrm{R}$ restrictions. This concludes the description of Algorithm 2.

We say that a *conflict* occurs when a tuple is to be stored in a cell $T_i[D][V]$ which is already occupied by another (non-empty) tuple. Note that conflicting tuples must have the same difference and validity vectors and only use elements up to i. Since the DP process must minimize the overall ratio, including potential singletons not participating in said process, conflict resolution becomes more involved for $k > 2$.

▶ **Lemma 10** (Conflict resolution). *Let $C_1 = (S_1, \ldots, S_k, sum_1)$ and $C_2 = (S'_1, \ldots, S'_k, sum'_1)$ be two tuples that result in a conflict in a cell $T_i[D][V]$ of the DP table of Algorithm 2, such that $sum_1 < sum'_1$. No optimal k-SSR$_\mathrm{R}$ solution may use*[8] *C_1.*

Proof. Let S be a k-SSR$_\mathrm{R}$ solution. Split S in two: S_{small}, containing the k' sets returned by Algorithm 2, and S_{large}, containing the singletons $\{j\}$ s.t. $a_j > Q$. Assume S_{small} uses C_1.

By assumption, C_1 and C_2 have identical D and V vectors and their sets involve only elements from $[i]$. This implies that any combination of elements larger than i that can later be added to sets of C_1 to construct a partial[9] solution with vectors D', V' can also be added to the corresponding sets of C_2 to construct a partial solution with the same vectors D', V'. Let S'_{small} be the partial solution obtained by using C_2 instead of C_1 and adding the same combination of elements that would have been used to obtain S_{small} from C_1. Consider the solution S' that is obtained by appending S_{large} to S'_{small}. Note that S_{small} and S'_{small} have the same validity vector, hence S' is also a feasible k-SSR$_\mathrm{R}$ solution.

---

[8] In our dynamic programming framework, we say that a solution S *uses* a tuple $C = (S_1, \ldots, S_k, sum_1)$ if C appears in an intermediate step of the construction of S.

[9] We denote as *partial* a solution for which S_{large} will be appended to obtain a feasible k-SSR$_\mathrm{R}$ solution.

Let M_s, m_s be the maximum and minimum sum of S_{small} respectively and M'_s, m'_s those of S'_{small}. Since $sum_1 < sum'_1$ and the difference vector D is the same for both solutions, it follows that $M_s < M'_s$, $m_s < m'_s$ and $M_s - m_s = M'_s - m'_s$. From these, we obtain

$$\frac{M_s}{m_s} > \frac{M'_s}{m'_s}. \tag{1}$$

Let $\{t\} \in S_{large}$ be the singleton set with the largest element. For all $\{i\} \in S_{large}$ such that $i \neq t$, we have

$$m_s \leq \Sigma(S_1, A) \leq Q < \Sigma(\{i\}, A) \leq \Sigma(\{t\}, A).$$

This implies that $\{i\}$ $(i \neq t)$ does not affect the ratio and $\{t\}$ only affects the ratio if $a_t > M_s$. Thus, all the sets in S_{large} can be ignored when calculating $\mathcal{R}(S, A)$, except for $\{t\}$ if $a_t > M_s$. The same holds for $\mathcal{R}(S', A)$ and M'_s. Consider three cases:

1. $M_s < M'_s < a_t$. In this case, $\mathcal{R}(S, A) = a_t/m_s$ and $\mathcal{R}(S', A) = a_t/m'_s$. Since $m_s < m'_s$: $\mathcal{R}(S, A) > \mathcal{R}(S', A)$.
2. $M_s < a_t \leq M'_s$. In this case, $\mathcal{R}(S, A) = a_t/m_s > M_s/m_s$ and $\mathcal{R}(S', A) = M'_s/m'_s$. By inequality (1): $\mathcal{R}(S, A) > \mathcal{R}(S', A)$.
3. $a_t \leq M_s < M'_s$. In this case, $\mathcal{R}(S, A) = M_s/m_s$ and $\mathcal{R}(S', A) = M'_s/m'_s$. By inequality (1): $\mathcal{R}(S, A) > \mathcal{R}(S', A)$.

In all cases $\mathcal{R}(S, A) > \mathcal{R}(S', A)$, therefore S cannot be optimal. ◀

▶ **Lemma 11** (Feasibility). *Every k-tuple of sets considered by Algorithm 1 is a feasible k-SSR$_{\mathrm{R}}$ solution.*

Proof. The x singletons constructed by Algorithm 1 are disjoint and contain elements larger than q. In Algorithm 2, every element $i \leq q$ is processed in increasing order and added to at most one set at a time. As such, all sets are disjoint. Note that p is contained in S_1 and S_1 cannot receive a larger element. Recall that v_i is set to `true` when S_i receives an element $j > p$ and Algorithm 2 only returns solutions with $V = [\texttt{true}, \ldots, \texttt{true}]$. Thus, all k-SSR$_{\mathrm{R}}$ restrictions regarding the largest element of each set are satisfied. ◀

The proof of the following theorem uses Lemmas 10 and 11. The main idea is to show that Algorithm 1 considers and therefore finds the optimal solution whose existence is guaranteed by Theorem 9.

▶ **Theorem 12** (Optimality). *Algorithm 1 returns an optimal solution for k-SSR$_{\mathrm{R}}$.*

Proof. Lemma 11 guarantees that Algorithm 1 returns a feasible k-SSR$_{\mathrm{R}}$ solution, so we only need to prove that its ratio is optimal. Let S^* be the optimal k-SSR$_{\mathrm{R}}$ solution guaranteed by Theorem 9. We split S^* in two: S^*_{large}, which contains the singleton sets $\{j\}$ s.t. $a_j > Q$, and S^*_{small}, which contains the rest of the sets. Note that the number x of these singletons is bounded by $k-1$ or by the amount of sufficiently large elements, i.e. $n-q$. Thus, Algorithm 1 considers every S_{large} that satisfies properties 2 and 3 of Theorem 9, including S^*_{large}.

Recall that Algorithm 1 uses Algorithm 2 as a subroutine in order to obtain candidate partial solutions, to which S^*_{large} will be appended. We would like S^*_{small} to be contained in the solutions returned by Algorithm 2.

Algorithm 2 prunes a solution if a difference d_j would become lower than or equal to $-2Q$. Any S_{small} satisfying property 1 of Theorem 9 will not be affected by this. This includes S^*_{small}.

When there is a conflict in a cell $T_i[D][V]$ between two tuples with different sum_1 values, S^*_{small} cannot use the one with smaller sum_1 (by Lemma 10), since S^* is a optimal. When there is a conflict in a cell $T_i[D][V]$ between two tuples C_1, C_2 with equal sum_1 values, observe that both tuples have exactly the same sums (but different sets), use only elements up to i and have the same validity vector. This implies that any combination of elements greater than i that can later be added to sets of C_1 can also be added to the same sets of C_2 and vice versa. Thus, any combination of sums that can be reached through C_1 can also be reached through C_2 and vice versa. This means that if one of these tuples leads to S^*_{small}, the other one leads to another partial solution S^{**}_{small}, which has exactly the same sums as S^*_{small}. Appending S^*_{large} to S^{**}_{small} yields a solution with the same sums as S^*, i.e. another optimal solution.

It follows directly from the description of Algorithm 2 that it constructs every possible combination of disjoint sets $S_1, \ldots, S_{k-x}$ with $\max(S_1) = p$, apart from the ones pruned as explained in the previous paragraphs.

Therefore, the partial solutions returned by Algorithm 2 contain S^*_{small} or another partial solution with the same sums, which also leads to an optimal solution when appended with S^*_{large}. In either case, Algorithm 1 will find an optimal solution when iterating to find the best ratio. ◀

▶ **Lemma 13** (Complexity). *Algorithm* 1 *runs in time* $O(nQ^{k-1})$.

Proof. Algorithm 1 calls Algorithm 2 once for each value of x in $\{0, \ldots, \min\{k-1, n-q\}\}$, where x is the number of singleton sets $\{j\}$ s.t. $a_j > Q$. For a fixed x, Algorithm 2 is applied to an instance with $k' = k - x$ sets and $q = O(n)$ elements.

Due to the pruning done in Algorithm 2, it holds that $\forall j \in \{2, \ldots, k'\}$: $-2Q < d_j \leq Q$, where the second inequality holds because $\Sigma(S_1, A) \leq Q$. Since the algorithm stores D in non-decreasing order (i.e., $d_2 \leq d_3 \leq \cdots \leq d_{k'}$), there are $O((3Q)^{k'-1}/(k'-1)!)$ distinct difference vectors. Taking into account the validity vector V and all T_i's, we obtain the following bound for the amount of DP cells: $O(n(6Q)^{k'-1}/(k'-1)!)$.

For constant k, this bound becomes $O(nQ^{k'-1})$. Note that all operations of Algorithm 2 on DP cells (such as sorting the difference vector after adding an element) run in time $O(k')$, so the time complexity of Algorithm 2 is $O(nQ^{k'-1}) = O(nQ^{k-x-1})$. Hence, the time complexity of Algorithm 1 is bounded by:

$$\sum_{x=0}^{k-1} nQ^{k-x-1} = O(nQ^{k-1})$$ ◀

3.3 FPTASs for k-SSR_R and k-SSR

By Theorem 12 and Lemma 13, Algorithm 1 solves k-SSR_R in pseudo-polynomial time. To obtain an FPTAS for k-SSR_R, we scale (and round down) the input set by a factor $\delta = \frac{\varepsilon \cdot a_p}{3 \cdot n}$ (cf. [31, 35]).

This leads to Algorithm 3, which calls Algorithm 1 in order to find an optimal solution for the rounded input A_r, yielding a $(1+\varepsilon)$-approximation of the (largest-to-smallest) ratio of an optimal solution $(S^*_1, \ldots, S^*_k)$ of the original k-SSR_R instance.

Algorithm 3 FPTAS_k-SSR$_{\mathrm{R}}(A, p, \varepsilon)$.

Input: A sorted multiset $A = \{a_1, \ldots, a_n\}, a_i \in \mathbb{Z}^+$, an integer p such that $1 \leq p \leq n-k+1$ and an error parameter $\varepsilon \in (0, 1)$.

Output: Disjoint subsets $(S_1, \ldots, S_k)$ of $[n]$ with $\max(S_1) = p$, $\max(S_i) > p$ for $1 < i \leq k$ and $\mathcal{R}(S_1, \ldots, S_k, A) \leq (1+\varepsilon) \cdot \mathcal{R}(S_1^*, \ldots, S_k^*, A)$.

1: $\delta \leftarrow \frac{\varepsilon \cdot a_p}{3 \cdot n}$, $A_r \leftarrow \emptyset$
2: **for** $i \leftarrow 1, \ldots, n$ **do** $a_i^r \leftarrow \lfloor \frac{a_i}{\delta} \rfloor$, $A_r \leftarrow A_r \cup \{a_i^r\}$
3: $(S_1, \ldots, S_k) \leftarrow$ Exact_k-SSR$_{\mathrm{R}}(A_r, p)$ ▷ Call Alg. 1 for rounded instance
4: **return** $S_1, \ldots S_k$

▶ **Theorem 14** (k-SSR$_{\mathrm{R}}$ approximation). *Let $(S_1, \ldots, S_k)$ be the sets returned by Algorithm 3 for a k-SSR$_{\mathrm{R}}$ instance $(A = \{a_1, \ldots, a_n\}, p)$ with error parameter ε. Let $(S_1^*, \ldots, S_k^*)$ be an optimal solution for the same k-SSR$_{\mathrm{R}}$ instance. Then:*

$$\mathcal{R}(S_1, \ldots, S_k, A) \leq (1+\varepsilon) \cdot \mathcal{R}(S_1^*, \ldots, S_k^*, A)$$

The proof of Theorem 14 closely follows the proofs presented in section 5 of [31]. We include the proof in the full version of this paper [24].

By Lemma 13, Algorithm 3 solves k-SSR$_{\mathrm{R}}$ in $O(nQ^{k-1})$ (where $Q = \sum_{i=1}^{p} a_i^r$). Since we scaled the input by δ, the values of Q are bounded as follows:

$$Q^{k-1} = \left(\sum_{i=1}^{p} a_i^r\right)^{k-1} \leq \left(n \cdot a_p^r\right)^{k-1} \leq \left(\frac{n \cdot a_p}{\delta}\right)^{k-1} = \left(\frac{3 \cdot n^2}{\varepsilon}\right)^{k-1} = \frac{3^{k-1} \cdot n^{2k-2}}{\varepsilon^{k-1}}$$

Therefore, Algorithm 3 runs in time $O(n^{2k-1}/\varepsilon^{k-1})$. To obtain an FPTAS for the (unrestricted) k-SSR problem, we run Algorithm 3 once for each possible value of p ($1 \leq p \leq n-k+1$) and pick the solution with the best ratio. Thus, we obtain the main theorem of this section.

▶ **Theorem 15.** *There is an FPTAS for k-SSR that runs in $O(n^{2k}/\varepsilon^{k-1})$ time.*

4 An FPTAS for k-way Number Partitioning Ratio

In this section we present an FPTAS for k-way Number Partitioning Ratio (k-PART) by extending and refining the techniques of Section 3. Recall that in the case of k-PART, every element needs to be assigned to a set. We define the following restricted version of k-PART, which is analogous to k-SSR$_{\mathrm{R}}$.

▶ **Definition 16** (k-PART$_{\mathrm{R}}$). *Given a sorted multiset $A = \{a_1, \ldots, a_n\}$ of positive integers and an integer $p, 1 \leq p \leq n-k+1$, find k disjoint subsets $S_1, \ldots, S_k$ of $[n]$ with $\bigcup_{i=1}^{k} S_i = [n]$, $\max(S_1) = p$ and $\max(S_i) > p$ for $1 < i \leq k$, such that $\mathcal{R}(S_1, \ldots, S_k, A)$ is minimized.*

The main challenge in expanding our technique to k-PART stems from the fact that there exist instances (A, p) of k-PART$_{\mathrm{R}}$ where all optimal solutions violate one or more of the properties of Theorem 9. This is a problem, since the time complexity of our k-SSR$_{\mathrm{R}}$ algorithm relies heavily on Theorem 9.

We overcome this by showing that there exists a p^* such that the corresponding optimal solution S^* to (A, p^*) is well-behaved (see Theorem 18). It suffices to guarantee that for p^*, the algorithm will consider S^* (or another equivalent solution) as a candidate solution for the rounded k-PART$_{\mathrm{R}}$ instance. Note that since S^* is not guaranteed to be optimal for the rounded k-PART$_{\mathrm{R}}$ instance, we obtain *neither an exact algorithm nor an FPTAS* for the restricted problem k-PART$_{\mathrm{R}}$.

4.1 Properties of k-PART$_{\mathrm{R}}$ instances

▶ **Definition 17** (Perfect p)**.** *Let $A = \{a_1, \ldots, a_n\}$ be the input to the k-PART problem. We call a number $p : 1 \leq p \leq n - k + 1$* perfect *for A if the optimal solution(s) for the k-PART$_{\mathrm{R}}$ instance (A, p) have the same (largest-to-smallest) ratio as the optimal solution(s) for the k-PART instance A.*

We define $Q = \sum_{i=1}^{p} a_i$ and $q = \max\{i \mid a_i \leq Q\}$, in the same manner as we did for k-SSR$_{\mathrm{R}}$. The next theorem is analogous to Theorem 9, with a key difference: its properties are only guaranteed for some *perfect* p. Its proof uses arguments similar in spirit to the proof of Theorem 9. Special attention is required since elements that violate a property are reassigned to other sets rather than being removed entirely from the solution. This may cause some k-PART$_{\mathrm{R}}$ restriction to be violated, thus rendering the proof significantly more involved.

▶ **Theorem 18.** *Given a k-PART instance A, there exists a perfect p for A, for which there is an optimal solution for the k-PART$_{\mathrm{R}}$ instance (A, p) satisfying the following properties:*
1. *For all sets S_i containing only elements $j \leq q$ it holds that $\Sigma(S_i, A) < 2Q$.*
2. *All elements $j > q$ are contained in singleton sets.*

Proof. For some arbitrary value of p, let $S = (S_1, \ldots, S_k)$ be an arbitrary feasible solution for the k-PART$_{\mathrm{R}}$ instance (A, p). If S breaks property 2, i.e. there exists a set S_i in S with $|S_i| > 1$ containing an element $j > q$, we do the following. Let S_m be a set with minimum sum[10] and u the smallest element of S_i. The following inequalities hold.

$$m(S, A) = \Sigma(S_m, A) \leq Q < \Sigma(S_i \setminus \{u\}, A)$$

$$\Sigma(S_m \cup \{u\}, A) \leq Q + a_u < \Sigma(S_i, A) \leq M(S, A)$$

From these we can infer that, by moving u from S_i to S_m, the minimum sum cannot decrease and the maximum cannot increase. Thus, we move u as described, yielding a solution S' with $\mathcal{R}(S', A) \leq \mathcal{R}(S, A)$. However, it might be the case that S_m is S_1 and adding u to it breaks the restrictions of k-PART$_{\mathrm{R}}$, so S' might not be a feasible solution for (A, p). For the new solution S', define p' as the minimum among the maxima of its k sets and call S_1 the set that contains p'. Note that S' is a feasible solution for the k-PART$_{\mathrm{R}}$ instance (A, p').

If S' still breaks property 2 for $Q' = \sum_{i=1}^{p'} a_i$ and $q' = \max\{i \mid a_i \leq Q'\}$, apply the same process. By doing this repeatedly, p cannot decrease and the ratio of the solution cannot increase. Note that p cannot exceed $n - k + 1$ with this process, so at some point p will stop increasing. Let p_f be this final value and define Q_f, q_f accordingly for p_f. Repeating the process will at some point yield a solution that satisfies property 2 for the k-PART$_{\mathrm{R}}$ instance (A, p_f).

Let S'' be the feasible solution for the k-PART$_{\mathrm{R}}$ instance (A, p_f), derived from the process described in the previous paragraphs. S'' satisfies property 2 for values Q_f, q_f. If S'' breaks property 1, i.e. it contains a set S_i consisting only of elements $j \leq q_f$ and having sum $\Sigma(S_i, A) \geq 2Q_f$, we apply the following transformation. Let S_m be a set with minimum sum. Take the smallest element j from S_i and move it to S_m. Observe that $a_j \leq Q_f$. Hence, the following inequalities hold.

$$m(S'', A) = \Sigma(S_m, A) \leq Q_f \leq \Sigma(S_i \setminus \{j\}, A)$$

$$\Sigma(S_m \cup \{j\}, A) \leq 2Q_f \leq \Sigma(S_i, A) \leq M(S'', A)$$

[10] There could be multiple sets with the same (minimum) sum.

From these we can infer that, by moving j from S_i to S_m, the minimum sum cannot decrease and the maximum cannot increase. We repeatedly apply this process until we end up with a solution satisfying property 1, just like we did in the previous paragraphs for property 2, since the same arguments hold regarding the increment of p and the ratio not increasing. Furthermore, this transformation preserves property 2, as p cannot decrease through this process and we only add elements to the set with the smallest sum (which cannot be a set containing an element $j > q$).

In conclusion, for any feasible solution for a k-PART$_{\mathrm{R}}$ instance (A, p) that violates some property, we can find a feasible solution for another k-PART$_{\mathrm{R}}$ instance (A, p_{new}) that satisfies both properties and has equal or smaller ratio. Suppose we apply this to an optimal solution for a k-PART$_{\mathrm{R}}$ instance (A, p), with p being *perfect*[11] for the k-PART instance A. Since the ratio of the solution does not increase throughout the constructions described in this proof, any new p' obtained by the constructions must also be *perfect* and the solution obtained must be optimal for (A, p'). ◀

Let p^* be one of the *perfect* elements guaranteed to exist by Theorem 18 for A and S^* be a respective optimal solution for (A, p^*) satisfying properties 1 and 2. The following lemmas indicate that we can prune certain values of p, without missing S^*. We define $x = n - q$. Intuitively, for (A, p^*), x is the number of large elements that must be contained in singleton sets in S^*, according to Theorem 18.

▶ **Lemma 19.** *If for a k-PART$_{\mathrm{R}}$ instance (A, p) we have $x > k - 1$, then $p \neq p^*$.*

Proof. Since S_1 cannot contain elements $j > q$, there are $k - 1$ sets that can contain such elements. The number of these elements is x. If $x > k - 1$, by the pigeonhole principle, every feasible solution contains a set with two or more of these elements. This contradicts property 2 of Theorem 18, therefore $p \neq p^*$. ◀

▶ **Lemma 20.** *If for a k-PART$_{\mathrm{R}}$ instance (A, p) we have $x = k - 1$ and $p < q$, then $p \neq p^*$.*

Proof. Because $p < q$, S_1 cannot contain q. Thus, there are at least $x + 1 = k$ elements that cannot be contained in S_1, with $k - 1$ of them being greater than q. By the pigeonhole principle, every feasible solution contains a set with two or more elements, one of which is greater than q. This contradicts property 2 of Theorem 18, therefore $p \neq p^*$. ◀

4.2 Obtaining an FPTAS for *k*-PART

We now present the main algorithm for k-PART. Its design is analogous to that of Algorithms 1 and 3, with two important differences:

1. There is no iteration for different amounts of singletons with elements $j > q$. Instead, we prune some values of p according to Lemmas 19 and 20 and fix x singletons for the rest.
2. In contrast to k-SSR$_{\mathrm{R}}$, we do not necessarily obtain the optimal solution to each rounded k-PART$_{\mathrm{R}}$ instance (A_r, p). Interestingly, we will later prove that the obtained solution is sufficient for the algorithm to be an FPTAS for k-PART.

[11] Such a p always exists, by definition.

Algorithm 4 FPTAS_k-PART(A, ε).

Input: A sorted multiset $A = \{a_1, \ldots, a_n\}, a_i \in \mathbb{Z}^+$ and an error parameter $\varepsilon \in (0, 1)$.
Output: Disjoint subsets $(S_1, \ldots, S_k)$ of $[n]$, such that $\bigcup_{i=1}^{k} S_i = [n]$ and their ratio satisfies $\mathcal{R}(S_1, \ldots, S_k, A) \leq (1+\varepsilon)\mathcal{R}(S_1^*, \ldots, S_k^*, A)$.

```
for p ← 1, ..., n − k + 1 do
    best_ratio[p] ← ∞, best_solution[p] ← (∅, ..., ∅)
for p ← 1, ..., n − k + 1 do
    Q ← Σ_{i=1}^{p} a_i, q ← max{i | a_i ≤ Q}, x ← n − q
    if x > k − 1 ∨ (x = k − 1 ∧ p < q) then                ▷ Lemmas 19 and 20
        continue to next p
    for y ← 1, ..., x do S_{k−x+y} ← {q + y}               ▷ x singletons
    δ ← ε·a_p / (3·n), A_r ← ∅, k' ← k − x
    for i ← 1, ..., n do a_i^r ← ⌊a_i / δ⌋, A_r ← A_r ∪ {a_i^r}
    A'_r ← {a_1^r, ..., a_q^r}                             ▷ q items for DP
    DP_solutions ← DP_k-PART_R(A'_r, k', p, Q/δ)
    for all (S_1, ..., S_{k'}) in DP_solutions do
        current_ratio ← R(S_1, ..., S_k, A_r)              ▷ k-PART_R solution
        if current_ratio < best_ratio[p] then
            best_solution[p] ← (S_1, ..., S_k)             ▷ Best solution for each A_r
            best_ratio[p] ← current_ratio                  ▷ and its ratio
final_ratio ← ∞, final_solution ← 0
for p ← 1, ..., n − k + 1 do                               ▷ Iterate for all p to find best sol. for A
    current_ratio ← R(best_solution[p], A)
    if current_ratio < final_ratio then
        final_solution ← best_solution[p]
        final_ratio ← current_ratio
return final_solution
```

Algorithm 4 calls a dynamic programming subroutine for each value of p, in order to find candidate partial solutions for elements $j \leq q$. This DP subroutine, DP_k-PART$_{\mathrm{R}}(A'_r, k', p, Q)$, is a direct extension of Algorithm 2, with the following three differences.

1. The case of an element not being added to any set is skipped.
2. When a conflict occurs in a DP cell $T_i[D][V]$, we do not use sum_1 to resolve it. For k-PART$_{\mathrm{R}}$ every element is included in some set, thus conflicts can only occur between tuples with identical sums. Hence, it does not matter which tuple is preferred (as proven in Theorem 12 for conflicts with equal sums) and there is no need to ever consider sum_1.
3. The bound for pruning large negative differences is chosen as $-2Q/\delta$, where $Q = \sum_{i=1}^{p} a_i$ is calculated using the *initial* values a_i instead of the scaled and rounded values a_i^r. This is necessary to avoid pruning edge case solutions. We will expand on this in Lemma 22.

The respective pseudocode is included in the full version of this work [24]. We now present the following lemma, whose proof is essentially identical to that of Lemma 11.

▶ **Lemma 21** (Feasibility). *Every k-tuple of sets whose $\mathcal{R}(S_1, \ldots, S_k, A_r)$ value is considered by Algorithm 4 is a feasible solution for the k-PART$_{\mathrm{R}}$ instance (A_r, p).*

Recall that we defined p^* and S^* as a *perfect* p and a respective optimal solution for (A, p^*) whose existence is guaranteed by Theorem 18.

▶ **Lemma 22** (Near-optimality). *When Algorithm 4 iterates to compare the ratio of solutions for the k-PART$_{\mathrm{R}}$ instance (A_r, p^*), it will consider either S^* or another solution $S = (S_1, \ldots, S_k)$ with $\Sigma(S_i, A_r) = \Sigma(S_i^*, A_r)$, $\forall i \in [k]$.*

Proof. According to Lemmas 19 and 20, we cannot have $x > k-1$ or $x = k-1$ and $p < q$ for $p = p^*$. The singleton sets enforced by Algorithm 4 are exactly the singleton sets contained in S^*, according to property 2 of Theorem 18.

Recall that a solution in the DP subroutine is pruned if it has some difference $d_j \leq -2Q/\delta$. By property 1 of Theorem 18, it holds that $\Sigma(S_i^*, A) < 2Q$, $\forall i \in [k-x]$. Since A_r contains elements scaled by δ and *rounded down*, we infer that $\forall i \in [k-x]$

$$\Sigma(S_i^*, A_r) \leq \Sigma(S_i^*, A)/\delta < 2Q/\delta.$$

This implies that for S^*, there will be no $d_j \leq -2Q/\delta$ (at any point in its dynamic programming construction).

As already explained, DP conflicts for k-PART$_{\mathrm{R}}$ occur only between tuples with identical sums. Recall that two conflicting tuples have the same vectors D, V and only use elements up to some element i, so any combination of elements that can be added to sets of one tuple can also be added to the respective sets of the other tuple. This is explained in more detail in the proof of Theorem 12. We infer that it is possible for S^* to be overwritten by another solution $S = (S_1, \ldots, S_k)$ with $\Sigma(S_i, A_r) = \Sigma(S_i^*, A_r)$, $\forall i \in [k]$.

The DP subroutine constructs every possible combination of disjoint sets $S_1, \ldots, S_{k-x}$ with $\max(S_1) = p$ and $\bigcup_{i=1}^{k} S_i = [q]$, apart from the ones pruned by the cases mentioned in the previous paragraph. Thus, the lemma follows. ◀

▶ **Theorem 23.** *Algorithm 4 is an FPTAS for k-PART that runs in time $O(n^{2k}/\varepsilon^{k-1})$.*

Proof. We rely upon the observation that the proof of Theorem 14 does not necessarily require an optimal solution for the rounded k-SSR$_{\mathrm{R}}$ instance A_r; it suffices for the algorithm to consider a k-SSR$_{\mathrm{R}}$ solution S with $\mathcal{R}(S, A_r) \leq \mathcal{R}(S_{opt}, A_r)$, where S_{opt} is an optimal solution for the k-SSR instance A (prior to rounding).

First, consider the case $p \neq p^*$. If the conditions of Lemma 19 or 20 are met, this p will be skipped. Otherwise, a k-tuple of sets will be found, which is a feasible solution for the k-PART$_{\mathrm{R}}$ instance (A_r, p), according to Lemma 21.

Second, consider the case $p = p^*$. By Lemma 22, Algorithm 4 will consider S^* or another solution S with $\mathcal{R}(S, A_r) = \mathcal{R}(S^*, A_r)$ as a solution for the instance (A_r, p^*). Therefore, for the solution S_{alg} found by the algorithm in this iteration, it holds that $\mathcal{R}(S_{alg}, A_r) \leq \mathcal{R}(S^*, A_r)$. With the aforementioned observation, the proof of the following inequality is identical to the proof of Theorem 14.

$$\mathcal{R}(S_{alg}, A) \leq (1+\varepsilon)\mathcal{R}(S^*, A)$$

Taking into account both cases, the final solution returned by the algorithm is a feasible k-PART solution for A and has ratio smaller than or equal to that of the solution found in the p^*-th iteration. It follows that Algorithm 4 is a $(1+\varepsilon)$-approximation for k-PART.

We now analyze the complexity of the algorithm. The DP subroutine runs in time $O(n(Q/\delta)^{k-1})$, by the same reasoning as in the proof of Lemma 13 for $x = 0$ (which is the worst case). Taking into account the iteration for all possible values of p and using a bound for Q/δ just like we did for k-SSR, we infer that Algorithm 4 runs in $O(n^{2k}/\varepsilon^{k-1})$. ◀

5 An FPTAS for k-SSR with linear dependence on n

In this section, we provide an alternative FPTAS for k-SSR, using techniques inspired from [8] to reduce the problem to smaller instances. We then use the k-SSR FPTAS of Section 3 to solve them.

We define an alternative restricted version of k-SSR, called k-SSR$_{\mathrm{L}}$, in which the solution is forced to contain the largest element of the input. This definition is based on the SSR$_{\mathrm{L}}$ problem defined in [8].

▶ **Definition 24** (k-SSR$_{\mathrm{L}}$). *Given a sorted multiset $A = \{a_1, \ldots, a_n\}$ of positive integers, find k disjoint subsets $S_1, \ldots, S_k$ of $[n]$ with $n \in \bigcup_{i=1}^{k} S_i$, such that $\mathcal{R}(S_1, \ldots, S_k, A)$ is minimized.*

Let $A = \{a_1, \ldots, a_n\}$ be a sorted multiset of n positive integers. We denote by $A[l, r]$ the subset of A consisting of all items a_i, such that $l \leq i \leq r$. We also use the notation $\mathrm{sum}(A) = \sum_{a \in A} a$.

For a multiset $A = \{a_1, \ldots, a_n\}$, we denote the optimal ratio of the k-SSR instance A as $\mathrm{OPT}(A)$ and the optimal ratio of the k-SSR$_{\mathrm{L}}$ instance A as $\mathrm{OPT}_{\mathrm{L}}(A)$. By definition, it holds that $1 \leq \mathrm{OPT}(A) \leq \mathrm{OPT}_{\mathrm{L}}(A)$.

▶ **Lemma 25.** *Let $A = \{a_1, \ldots, a_n\}$ be a sorted multiset of positive integers. It holds that*

$$\mathrm{OPT}(A) = \min_{k \leq j \leq n} \mathrm{OPT}_{\mathrm{L}}\left(A[1, j]\right).$$

Proof. Let $t \in \{k, \ldots, n\}$ be the maximum element[12] contained in some optimal solution S of the k-SSR instance A. By Definition 24, S is a feasible solution for the k-SSR$_{\mathrm{L}}$ instance $A[1, t]$. This implies $\mathrm{OPT}(A) \geq \min_{k \leq j \leq n} \mathrm{OPT}_{\mathrm{L}}\left(A[1, j]\right)$.

Note that any feasible solution for a k-SSR$_{\mathrm{L}}$ instance $A[1, j]$ (for any $j \in \{k, \ldots, n\}$) is a feasible solution for the k-SSR instance A. Thus, it also holds that $\mathrm{OPT}(A) \leq \min_{k \leq j \leq n} \mathrm{OPT}_{\mathrm{L}}\left(A[1, j]\right)$. ◀

Lemma 26 provides the key structural insight of our algorithm. Intuitively, the fact that a k-SSR$_{\mathrm{L}}$ solution contains the largest element allows us to consider only the largest $O\big((1/\varepsilon)\ln(1/\varepsilon)\big)$ elements for each k-SSR$_{\mathrm{L}}$ instance $A[1, j]$, in order to obtain a $(1+\varepsilon)$-approximation of the optimal k-SSR solution.

▶ **Lemma 26** (Largest elements). *For any sorted multiset $A = \{a_1, \ldots, a_n\}$ of positive integers and any $\varepsilon \in (0, 1)$, the following holds:*

$$\mathrm{OPT}(A) \leq \min_{k \leq j \leq n} \mathrm{OPT}_{\mathrm{L}}\left(A\big[j - C + 1, j\big]\right) \leq (1+\varepsilon)\mathrm{OPT}(A), \tag{2}$$

where $C = (c+1)(k-1)$ and $c = 1 + \left\lceil \left(1 + \frac{1}{\varepsilon}\right) \ln \frac{2(k-1)}{\varepsilon^2} \right\rceil$.

Proof. Since $A\big[j - C + 1, j\big]$ is a subset of A and k-SSR$_{\mathrm{L}}$ is (by definition) a restricted version of k-SSR, it follows directly that

$$\mathrm{OPT}(A) \leq \mathrm{OPT}\left(A\big[j - C + 1, j\big]\right) \leq \mathrm{OPT}_{\mathrm{L}}\left(A\big[j - C + 1, j\big]\right).$$

Note that the above is true for any $j \in [n]$, therefore it is also true for the minimum. Thus, the first inequality of (2) holds. As for the second inequality of (2), we distinguish between two cases.

[12] If $t < k$, all feasible solutions for both problems have infinite ratio.

Case 1 (Dense case): There exists some $v \in \{k, \ldots, n\}$ s.t. $a_v \leq (1+\varepsilon)a_{v-k+1}$. Since $C \geq k$, by setting $S_i = \{v-k+i\}$ for each $i \in [k]$, we obtain

$$\min_{k \leq j \leq n} \mathrm{OPT}_{\mathrm{L}}(A[j-C+1, j]) \leq \mathrm{OPT}_{\mathrm{L}}(A[v-C+1, v]) \leq (1+\varepsilon) \leq (1+\varepsilon)\mathrm{OPT}(A).$$

Case 2 (Sparse case): For all $v \in \{k, \ldots, n\}$ it holds that $a_v > (1+\varepsilon)a_{v-k+1}$. By repeatedly applying this inequality, it follows that for all $v \in \{k, \ldots, n\}$ and all $i \geq 1$ such that $v - i(k-1) \geq 1$,

$$a_{v-i(k-1)} < (1+\varepsilon)^{-i} a_v. \tag{3}$$

Thus, for all such v, i it holds that

$$\begin{aligned} \mathrm{sum}\Big(A\big[v-(i+1)(k-1)+1,\ v-i(k-1)\big]\Big) &\leq (k-1)a_{v-i(k-1)} \\ &< (k-1)(1+\varepsilon)^{-i}a_v. \quad \text{[by Ineq. (3)]} \end{aligned} \tag{4}$$

Consider the set $A\big[1, v-c(k-1)\big]$. Splitting this set into subsets of size (at most) $k-1$ and using (4) for each of them yields

$$\mathrm{sum}\Big(A\big[1, v-c(k-1)\big]\Big) \leq (k-1)\sum_{i=0}^{\infty}(1+\varepsilon)^{-(c+i)}a_v = \frac{(k-1)a_v}{\varepsilon(1+\varepsilon)^{c-1}}, \tag{5}$$

where the last step is calculated as the sum of a geometric series. Note that (5) holds even if $v - c(k-1) < 1$, since then it would be $\mathrm{sum}\Big(A\big[1, v-c(k-1)\big]\Big) = 0$. Substituting c into (5) and using $(1+\varepsilon)^{1+1/\varepsilon} > e$ yields

$$\mathrm{sum}\Big(A\big[1, v-c(k-1)\big]\Big) < \frac{\varepsilon}{2} \cdot a_v.$$

For any $k \leq j \leq n$, we choose $v = j-k+1$ to obtain

$$\mathrm{sum}\Big(A\big[1, j-C\big]\Big) < \frac{\varepsilon}{2} \cdot a_{j-k+1}. \tag{6}$$

Intuitively, inequality (6) is a bound for the sum of the $j-C$ *smallest* elements of A. This inequality will be used to obtain a $(1+\varepsilon)$-approximation by removing these elements from an optimal k-$\mathrm{SSR}_{\mathrm{L}}$ solution.

Let $S = (S_1, \ldots, S_k)$ be an optimal solution for the k-$\mathrm{SSR}_{\mathrm{L}}$ instance $A_j = A[1, j]$. We define

$$S_M = \arg\max_{S_i \in S} \Sigma(S_i, A_j) \quad \text{and} \quad S_m = \arg\min_{S_i \in S} \Sigma(S_i, A_j).$$

Now consider the k-$\mathrm{SSR}_{\mathrm{L}}$ instance $A'_j = A\big[j-C+1, j\big]$ (i.e. A'_j contains the C *largest* elements of A_j). For each $S_i \in S$, we define $S'_i = S_i \cap [j-C+1, j]$, i.e. the subset of S_i that contains elements i such that $a_i \in A'_j$. We also define the respective k-tuple of sets, $S' = (S'_1, \ldots, S'_k)$, and the following sets,

$$S_{\mathcal{M}} = \arg\max_{S_i \in S} \Sigma(S'_i, A'_j) \quad \text{and} \quad S_{\mu} = \arg\min_{S_i \in S} \Sigma(S'_i, A'_j).$$

Observe that, by the above definitions, $S'_{\mathcal{M}}$ is a set with maximum sum among all sets in S' and S'_{μ} is a set with minimum sum among all sets in S'. By (6), for all $i \in [k]$ it holds that

$$\Sigma(S_i, A_j) - \Sigma(S'_i, A'_j) \leq \mathrm{sum}\Big(A\big[1, j-C\big]\Big) < \frac{\varepsilon}{2} \cdot a_{j-k+1}.$$

Therefore, we obtain

$$\Sigma(S'_i, A'_j) \leq \Sigma(S_i, A_j) \leq \Sigma(S'_i, A'_j) + (\varepsilon/2) \cdot a_{j-k+1},\ \forall i \in [k]. \tag{7}$$

We now obtain a bound for the ratio $\mathcal{R}(S', A'_j)$ as follows.

$$\begin{aligned}\mathcal{R}(S', A'_j) = \frac{\Sigma\left(S'_{\mathcal{M}}, A'_j\right)}{\Sigma\left(S'_{\mu}, A'_j\right)} &\leq \frac{\Sigma\left(S_{\mathcal{M}}, A_j\right)}{\Sigma\left(S_{\mu}, A_j\right) - \frac{\varepsilon}{2} \cdot a_{j-k+1}} \qquad \text{[by Ineq. (7)]} \\ &\leq \frac{\Sigma\left(S_M, A_j\right)}{\Sigma\left(S_m, A_j\right) - \frac{\varepsilon}{2} \cdot a_{j-k+1}}. \end{aligned} \tag{8}$$

Recall that $j \in \bigcup_{i=1}^{k} S_i$ (by Definition 24), which implies that $\Sigma(S_M, A_j) \geq a_j$. Let $S^s = \{S_1^s, \ldots, S_k^s\}$ be a solution consisting of k singleton sets that contain the k largest elements of A_j, i.e. $S_i^s = \{j - i + 1\}$, $\forall i \in [k]$. Since S^s is a feasible solution for k-SSR$_{\mathrm{L}}$ with ratio a_j/a_{j-k+1}, it holds that $\mathcal{R}(S, A_j) = \Sigma(S_M, A_j)/\Sigma(S_m, A_j) \leq a_j/a_{j-k+1}$. Thus, we have

$$\Sigma(S_m, A_j) \geq \Sigma(S_M, A_j) \cdot \frac{a_{j-k+1}}{a_j} \geq a_{j-k+1}.$$

Combining this with (8), we obtain

$$\mathcal{R}(S', A'_j) \leq \frac{\Sigma(S_M, A_j)}{\Sigma(S_m, A_j)\left(1 - \frac{\varepsilon}{2}\right)} = \frac{2}{2 - \varepsilon} \cdot \mathcal{R}(S, A_j).$$

Assuming $\varepsilon \in (0, 1)$, this becomes $\mathcal{R}(S', A'_j) \leq (1 + \varepsilon)\mathcal{R}(S, A_j)$. Note that some set S'_i contains j, so S' is a feasible solution for the k-SSR$_{\mathrm{L}}$ instance A'_j, therefore

$$\mathrm{OPT}_{\mathrm{L}}(A'_j) \leq \mathcal{R}(S', A'_j) \leq (1 + \varepsilon)\mathcal{R}(S, A_j) = (1 + \varepsilon)\mathrm{OPT}_{\mathrm{L}}(A_j).$$

By the definitions of A_j, A'_j, we have the following for all $j \in \{k, \ldots, n\}$:

$$\mathrm{OPT}_{\mathrm{L}}\left(A\big[j - C + 1, j\big]\right) \leq (1 + \varepsilon)\mathrm{OPT}_{\mathrm{L}}(A[1, j]).$$

Taking the minimum over all $j \in \{k, \ldots, n\}$ and using Lemma 25, we have

$$\min_{k \leq j \leq n} \mathrm{OPT}_{\mathrm{L}}\left(A\big[j - C + 1, j\big]\right) \leq (1 + \varepsilon)\mathrm{OPT}(A). \qquad \blacktriangleleft$$

▶ **Lemma 27** (Reduction). *If there is a $(1+\varepsilon)$-approximation algorithm for k-SSR$_{\mathrm{L}}$ running in time $T_L(n, \varepsilon)$, then there is a $(1+\varepsilon)$-approximation algorithm for k-SSR running in time*

$$O\left(nT_L\left(\frac{9k}{\varepsilon}\ln\frac{k}{\varepsilon}, \frac{\varepsilon}{3}\right)\right).$$

Proof. Let A be a multiset of n positive integers $a_1 \leq \ldots \leq a_n$. For each $j \in \{k, \ldots, n\}$, consider $A'_j = A\big[j - C + 1, j\big]$, where

$$C = (c + 1)(k - 1) \qquad \text{and} \qquad c = 1 + \left\lceil \left(1 + \frac{1}{\varepsilon}\right) \ln \frac{2(k-1)}{\varepsilon^2} \right\rceil.$$

Using the given $(1+\varepsilon)$-approximation algorithm for each of the k-SSR$_{\mathrm{L}}$ instances A'_j, we receive a k-tuple of sets $S^j = (S_1^j, \ldots, S_k^j)$ for which $\mathcal{R}\left(S^j, A'_j\right) \leq (1 + \varepsilon)\mathrm{OPT}_{\mathrm{L}}\left(A'_j\right)$. Thus,

$$\min_{k \leq j \leq n} \mathcal{R}\left(S^j, A'_j\right) \leq (1 + \varepsilon) \min_{k \leq j \leq n} \mathrm{OPT}_{\mathrm{L}}\left(A'_j\right) \leq (1 + \varepsilon)^2 \mathrm{OPT}(A). \qquad \text{[by Lemma 26]}$$

For all $j \in \{k, \ldots, n\}$, the set A'_j contains at most C elements, with

$$C = (k-1)\left(2 + \left\lceil \left(1+\frac{1}{\varepsilon}\right)\ln\frac{2(k-1)}{\varepsilon^2}\right\rceil\right) < 3k + k\left(1+\frac{1}{\varepsilon}\right)\ln\frac{2k}{\varepsilon^2} < \frac{9k}{\varepsilon}\ln\frac{k}{\varepsilon},$$

assuming $k \geq 2$ and $\varepsilon \in (0,1)$. As such, we have to run the given algorithm on $O(n)$ k-SSR$_{\mathrm{L}}$ instances of size bounded by $(9k/\varepsilon)\ln(k/\varepsilon)$ in order to obtain a $(1+\varepsilon)^2$-approximation for $\mathrm{OPT}(A)$. For all $\varepsilon \in (0,1)$ it holds that $(1+\varepsilon)^2 \leq 1+3\varepsilon$. Therefore, by setting the error parameter to $\varepsilon/3$ instead of ε in the approximation algorithm for k-SSR$_{\mathrm{L}}$, we obtain a $(1+\varepsilon)$-approximation for $\mathrm{OPT}(A)$ running in time

$$O\left(nT_L\left(\frac{9k}{\varepsilon}\ln\frac{k}{\varepsilon}, \frac{\varepsilon}{3}\right)\right). \qquad \blacktriangleleft$$

We now present the main theorem of this section, which follows by using the FPTAS of Theorem 15 as the $(1+\varepsilon)$-approximation algorithm for k-SSR$_{\mathrm{L}}$ in Lemma 27.

▶ **Theorem 28.** *There is an FPTAS for* k-SSR *that runs in* $\widetilde{O}(n/\varepsilon^{3k-1})$ *time, where* $\widetilde{O}$ *hides* $\mathrm{polylog}(1/\varepsilon)$ *factors.*

Proof. Let A be a k-SSR instance. Suppose we use our k-SSR FPTAS running in time $T_L(n,\varepsilon) = O(n^{2k}/\varepsilon^{k-1})$ (see Theorem 15) as a subroutine to solve the k-SSR$_{\mathrm{L}}$ instances $A'_j = A\big[j-C+1, j\big]$, $k \leq j \leq n$. This subroutine serves as a $(1+\varepsilon)$-approximation for k-SSR$_{\mathrm{L}}$, since for the solution $S^j = (S_1^j, \ldots, S_k^j)$ returned it holds that

$$\mathcal{R}\left(S^j, A'_j\right) \leq (1+\varepsilon)\mathrm{OPT}(A'_j) \leq (1+\varepsilon)\mathrm{OPT}_{\mathrm{L}}(A'_j).$$

Note that this subroutine does not take into account the k-SSR$_{\mathrm{L}}$ restriction $j \in \bigcup_{i=1}^{k} S_i^j$, therefore it might return an invalid k-SSR$_{\mathrm{L}}$ solution for A'_j. However, all solutions returned are feasible for the (unrestricted) k-SSR instance A, therefore the solution which yields $\min_{k\leq j\leq n}\mathcal{R}\left(S^j, A'_j\right)$ is a feasible one. As such, by Lemma 27 we obtain another FPTAS for k-SSR running in time

$$O\left(nT_L\left(\frac{9k}{\varepsilon}\ln\frac{k}{\varepsilon}, \frac{\varepsilon}{3}\right)\right) = \widetilde{O}\left(\frac{n}{\varepsilon^{3k-1}}\right). \qquad \blacktriangleleft$$

References

1. Giannis Alonistiotis, Antonis Antonopoulos, Nikolaos Melissinos, Aris Pagourtzis, Stavros Petsalakis, and Manolis Vasilakis. Approximating subset sum ratio via partition computations. *Acta Informatica*, 61(2):101–113, 2024. doi:10.1007/S00236-023-00451-7.
2. Georgios Amanatidis, Aris Filos-Ratsikas, and Alkmini Sgouritsa. Pushing the frontier on approximate EFX allocations. In *Proceedings of the 25th ACM Conference on Economics and Computation*, EC '24, pages 1268–1286, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3670865.3673582.
3. Antonis Antonopoulos, Aris Pagourtzis, Stavros Petsalakis, and Manolis Vasilakis. Faster algorithms for k-subset sum and variations. *J. Comb. Optim.*, 45(1):24, 2023. doi:10.1007/S10878-022-00928-0.
4. Cristina Bazgan, Miklos Santha, and Zsolt Tuza. Efficient approximation algorithms for the SUBSET-SUMS EQUALITY problem. *J. Comput. Syst. Sci.*, 64(2):160–170, 2002. doi:10.1006/JCSS.2001.1784.
5. Richard E. Bellman. *Dynamic programming*. Princeton University Press, Princeton, NJ, 1957.

6 Samuel Bismuth, Vladislav Makarov, Erel Segal-Halevi, and Dana Shapira. Partitioning problems with splittings and interval targets. In Julián Mestre and Anthony Wirth, editors, *35th International Symposium on Algorithms and Computation, ISAAC 2024, December 8-11, 2024, Sydney, Australia*, volume 322 of *LIPIcs*, pages 12:1–12:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ISAAC.2024.12.

7 Karl Bringmann. A near-linear pseudopolynomial time algorithm for subset sum. 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 1073–1084. SIAM, 2017. doi:10.1137/1.9781611974782.69.

8 Karl Bringmann. Approximating subset sum ratio faster than subset sum. In *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 1260–1277. SIAM, 2024. doi:10.1137/1.9781611977912.50.

9 Karl Bringmann and Vasileios Nakos. A fine-grained perspective on approximating subset sum and partition. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 1797–1815. SIAM, 2021. doi:10.1137/1.9781611976465.108.

10 Eric Budish. The combinatorial assignment problem: approximate competitive equilibrium from equal incomes. In Moshe Dror and Greys Sosic, editors, *Proceedings of the Behavioral and Quantitative Game Theory - Conference on Future Directions, BQGT '10, Newport Beach, California, USA, May 14-16, 2010*, page 74:1. ACM, 2010. doi:10.1145/1807406.1807480.

11 Lin Chen, Jiayi Lian, Yuchen Mao, and Guochuan Zhang. Approximating partition in near-linear time. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, STOC 2024, pages 307–318, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3618260.3649727.

12 Lin Chen, Jiayi Lian, Yuchen Mao, and Guochuan Zhang. An improved pseudopolynomial time algorithm for subset sum. In *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024, Chicago, IL, USA, October 27-30, 2024*, pages 2202–2216. IEEE, 2024. doi:10.1109/FOCS61266.2024.00129.

13 Lin Chen, Jiayi Lian, Yuchen Mao, and Guochuan Zhang. A note on deterministic FPTAS for partition, 2025. doi:10.48550/arXiv.2501.12848.

14 Mark Cieliebak and Stephan J. Eidenbenz. Measurement errors make the partial digest problem NP-hard. In Martin Farach-Colton, editor, *LATIN 2004: Theoretical Informatics, 6th Latin American Symposium, Buenos Aires, Argentina, April 5-8, 2004, Proceedings*, volume 2976 of *Lecture Notes in Computer Science*, pages 379–390. Springer, 2004. doi:10.1007/978-3-540-24698-5_42.

15 Mark Cieliebak, Stephan J. Eidenbenz, and Paolo Penna. Noisy data make the partial digest problem NP-hard. In Gary Benson and Roderic D. M. Page, editors, *Algorithms in Bioinformatics, Third International Workshop, WABI 2003, Budapest, Hungary, September 15-20, 2003, Proceedings*, volume 2812 of *Lecture Notes in Computer Science*, pages 111–123. Springer, 2003. doi:10.1007/978-3-540-39763-2_9.

16 Ed Coffman and Michael Langston. A performance guarantee for the greedy set-partitioning algorithm. *Acta Informatica*, 21:409–415, November 1984. doi:10.1007/BF00264618.

17 Bryan Deuermeyer, Donald Friesen, and Michael Langston. Scheduling to maximize the minimum processor finish time in a multiprocessor system. *SIAM Journal on Algebraic and Discrete Methods*, 3, June 1982. doi:10.1137/0603019.

18 Paul Duetting, Michal Feldman, and Yarden Rashti. Succinct ambiguous contracts, 2025. doi:10.48550/arXiv.2503.02592.

19 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

20 George Gens and Eugene Levner. A fast approximation algorithm for the subset-sum problem. *INFOR: Information Systems and Operational Research*, 32(3):143–148, 1994. doi:10.1080/03155986.1994.11732245.

21 Theodore P. Hill. Partitioning General Probability Measures. *The Annals of Probability*, 15(2):804–813, 1987. doi:10.1214/aop/1176992173.

22 Dorit S. Hochbaum and David B. Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *J. ACM*, 34(1):144–162, January 1987. doi:10.1145/7531.7535.

23 Ce Jin and Hongxun Wu. A simple near-linear pseudopolynomial time randomized algorithm for subset sum. In Jeremy T. Fineman and Michael Mitzenmacher, editors, *2nd Symposium on Simplicity in Algorithms, SOSA 2019, January 8-9, 2019, San Diego, CA, USA*, volume 69 of *OASIcs*, pages 17:1–17:6. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/OASICS.SOSA.2019.17.

24 Sotiris Kanellopoulos, Giorgos Mitropoulos, Antonis Antonopoulos, Nikos Leonardos, Aris Pagourtzis, Christos Pergaminelis, Stavros Petsalakis, and Kanellos Tsitouras. Approximation schemes for k-subset sum ratio and k-way number partitioning ratio, 2025. doi:10.48550/arXiv.2503.18241.

25 Hans Kellerer, Renata Mansini, Ulrich Pferschy, and Maria Grazia Speranza. An efficient fully polynomial approximation scheme for the subset-sum problem. *J. Comput. Syst. Sci.*, 66(2):349–370, 2003. doi:10.1016/S0022-0000(03)00006-0.

26 Konstantinos Koiliaris and Chao Xu. Faster pseudopolynomial time algorithms for subset sum. *ACM Trans. Algorithms*, 15(3):40:1–40:20, 2019. doi:10.1145/3329863.

27 Richard E. Korf. Objective functions for multi-way number partitioning. In Ariel Felner and Nathan R. Sturtevant, editors, *Proceedings of the Third Annual Symposium on Combinatorial Search, SOCS 2010, Stone Mountain, Atlanta, Georgia, USA, July 8-10, 2010*, pages 71–72. AAAI Press, 2010. doi:10.1609/SOCS.V1I1.18172.

28 David Kurokawa, Ariel D. Procaccia, and Junxing Wang. Fair enough: Guaranteeing approximate maximin shares. *J. ACM*, 65(2):8:1–8:27, 2018. doi:10.1145/3140756.

29 Joseph Y-T. Leung. Bin packing with restricted piece sizes. *Information Processing Letters*, 31(3):145–149, 1989. doi:10.1016/0020-0190(89)90223-8.

30 Richard J. Lipton, Evangelos Markakis, Elchanan Mossel, and Amin Saberi. On approximately fair allocations of indivisible goods. In Jack S. Breese, Joan Feigenbaum, and Margo I. Seltzer, editors, *Proceedings 5th ACM Conference on Electronic Commerce (EC-2004), New York, NY, USA, May 17-20, 2004*, pages 125–131. ACM, 2004. doi:10.1145/988772.988792.

31 Nikolaos Melissinos and Aris Pagourtzis. A faster FPTAS for the subset-sums ratio problem. In Lusheng Wang and Daming Zhu, editors, *Computing and Combinatorics - 24th International Conference, COCOON 2018, Qing Dao, China, July 2-4, 2018, Proceedings*, volume 10976 of *Lecture Notes in Computer Science*, pages 602–614. Springer, 2018. doi:10.1007/978-3-319-94776-1_50.

32 Stephan Mertens. The easiest hard problem: Number partitioning. In Allon G. Percus, Gabriel Istrate, and Cristopher Moore, editors, *Computational Complexity and Statistical Physics*, Santa Fe Institute Studies in the Sciences of Complexity, pages 125–140. Oxford University Press, 2006.

33 Marcin Mucha, Jesper Nederlof, Jakub Pawlewicz, and Karol Wegrzycki. Equal-subset-sum faster than the meet-in-the-middle. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, volume 144 of *LIPIcs*, pages 73:1–73:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.ESA.2019.73.

34 Marcin Mucha, Karol Wegrzycki, and Michal Wlodarczyk. A subquadratic approximation scheme for partition. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 70–88. SIAM, 2019. doi:10.1137/1.9781611975482.5.

35 Danupon Nanongkai. Simple FPTAS for the subset-sums ratio problem. *Inf. Process. Lett.*, 113(19-21):750–753, 2013. doi:10.1016/J.IPL.2013.07.009.

36 Trung Thanh Nguyen and Jörg Rothe. Minimizing envy and maximizing average nash social welfare in the allocation of indivisible goods. *Discret. Appl. Math.*, 179:54–68, 2014. doi:10.1016/J.DAM.2014.09.010.

37 Christos H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *J. Comput. Syst. Sci.*, 48(3):498–532, 1994. doi:10.1016/S0022-0000(05)80063-7.

38 Diego Recalde, Daniel Severín, Ramiro Torres, and Polo Vaca. An exact approach for the balanced k-way partitioning problem with weight constraints and its application to sports team realignment. *J. Comb. Optim.*, 36(3):916–936, 2018. doi:10.1007/S10878-018-0254-1.

39 Sartaj Sahni. Algorithms for scheduling independent tasks. *J. ACM*, 23(1):116–127, 1976. doi:10.1145/321921.321934.

40 Ethan L. Schreiber, Richard E. Korf, and Michael D. Moffitt. Optimal multi-way number partitioning. *J. ACM*, 65(4):24:1–24:61, 2018. doi:10.1145/3184400.

41 Max Springer, Mohammad Taghi Hajiaghayi, and Hadi Yami. Almost envy-free allocations of indivisible goods or chores with entitlements. In *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence and Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence and Fourteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'24/IAAI'24/EAAI'24. AAAI Press, 2024. doi:10.1609/aaai.v38i9.28851.

42 Gerhard J. Woeginger. When does a dynamic programming formulation guarantee the existence of a fully polynomial time approximation scheme (FPTAS)? *INFORMS J. Comput.*, 12(1):57–74, 2000. doi:10.1287/IJOC.12.1.57.11901.

43 Gerhard J. Woeginger and Zhongliang Yu. On the equal-subset-sum problem. *Inf. Process. Lett.*, 42(6):299–302, 1992. doi:10.1016/0020-0190(92)90226-L.

44 Yair Zadok, Nadav Voloch, Noa Voloch-Bloch, and Maor Meir Hajaj. Multiple subset problem as an encryption scheme for communication. *CoRR*, abs/2401.09221, 2024. doi:10.48550/arXiv.2401.09221.

45 Houyu Zhou, Tianze Wei, Biaoshuai Tao, and Minming Li. Fair allocation of items in multiple regions. In *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence and Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence and Fourteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'24/IAAI'24/EAAI'24. AAAI Press, 2024. doi:10.1609/aaai.v38i9.28861.

BFS and Reverse Shortest Paths for Ball Intersection Graphs in Three and Higher Dimensions

Matthew J. Katz
Department of Computer Science, Ben-Gurion University, Beer Sheva, Israel

Rachel Saban
Department of Computer Science, Ben-Gurion University, Beer Sheva, Israel

Micha Sharir
School of Computer Science and AI, Tel Aviv University, Israel

Abstract

Let $\mathcal{B}$ be a collection of n arbitrary balls in $\mathbb{R}^3$, and let $G_0(\mathcal{B})$ be their intersection graph. We provide an algorithm for performing BFS on $G_0(\mathcal{B})$, which runs in $O^*(n^{4/3})$ time, where the $O^*(\cdot)$ notation hides subpolynomial factors. For $r \geq 0$, let $G_r(\mathcal{B})$ be the intersection graph of the set $\mathcal{B}_r = \{B + r \mid B \in \mathcal{B}\}$, where $B + r$ is the ball concentric with B whose radius is larger by r than the radius of B. We provide an efficient algorithm for the reverse shortest path (RSP) problem, where we are given two designated balls B_s, B_t of $\mathcal{B}$ and a parameter $0 < \lambda < n$, and seek the smallest value r^* for which $G_{r^*}(\mathcal{B})$ contains a path from B_s to B_t of at most λ edges. For the special case of congruent balls (equivalently, for points in $\mathbb{R}^3$), the algorithm runs in $O^*(n^{29/21}) \approx O^*(n^{1.381})$ time. For the general case, the algorithm runs in $O^*(n^{56/39}) \approx O^*(n^{1.436})$ time. We also extend the technique to handle other measures of expansion and higher dimensions.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Computational geometry, reverse shortest paths, breadth-first search, shrink-and-bifurcate, intersection graphs

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.45

Funding *Matthew J. Katz*: Work partially supported by Grant 2019715/CCF-20-08551 from the US-Israel Binational Science Foundation/US National Science Foundation, and by Grant 495/23 from the Israel Science Foundation.
Rachel Saban: Work partially supported by Grant 495/23 from the Israel Science Foundation.
Micha Sharir: Work partially supported by Grant 495/23 from the Israel Science Foundation.

1 Introduction

The reverse shortest path (RSP) problem has received considerable attention recently. In general, we are given a set P of n geometric objects, and a parameter $r \geq 0$. We define a graph $G_r(P)$ on P, whose edges are all the pairs (u, v) that satisfy some property, given by a predicate $\Pi(u, v; r)$, which is monotone in r, meaning that if $\Pi(u, v; r)$ holds then $\Pi(u, v; r')$ holds for all $r' > r$. That is, the graphs $G_r(P)$ are monotone increasing in r. We are given two designated objects $s, t \in P$ and a parameter $0 < \lambda < n$, and wish to find the smallest value r^* for which $G_{r^*}(P)$ contains a path from s to t with at most λ edges.

Among the simplest examples is the case where P is a set of n points in the plane, and $G_r(P) = \{(u, v) \mid |uv| \leq r\}$, under the Euclidean distance $|uv|$. This can also be interpreted as the intersection graph of the disks of radius $r/2$ centered at the points of P (the so-called unit-disk graph, with the unit being $r/2$). This case and many variants thereof are reviewed later in the introduction.

© Matthew J. Katz, Rachel Saban, and Micha Sharir;
licensed under Creative Commons License CC-BY 4.0

One generalization, studied here, is to the case of points in three dimensions. That is, $G_r(P)$ in this case is the intersection graph of n congruent balls of radius $r/2$ centered at the points of P. This is a special instance of the general problem studied in this work, where the balls have arbitrary radii. That is, the input consists of a set $\mathcal{B}$ of n balls in $\mathbb{R}^3$ of arbitrary radii, and r is a parameter that measures how the balls of $\mathcal{B}$ are expanded. The simplest case is where we increase the radius of each ball by adding r to it. Let $G_r(\mathcal{B})$ denote the intersection graph of the expanded balls, so $G_0(\mathcal{B})$ is the intersection graph of the original balls. Our technique also applies to any other well-behaved measure of expansion, e.g., multiplying each radius by $r \geq 1$, but for conciseness we will mainly stick to the additive measure of expansion, as just introduced, and discuss the general setup later in the paper.

A standard high-level approach to the RSP problem is to design a decision procedure which, with r as input, determines whether $G_r(\mathcal{B})$ contains a path from s to t of length at most λ. If it does then $r^* \leq r$, and if not then $r^* > r$. The decision procedure is typically implemented using Breadth-First Search (BFS) on $G_r(\mathcal{B})$. Once we have such a procedure, we can use binary search on r to home in on the correct value of r^*.

There are two major problems with this approach. First, the cost of a naïve implementation of BFS is linear in the number of edges of $G_r(\mathcal{B})$, which can be $\Theta(n^2)$ in the worst case. Second, BFS appears not to be parallelizable, and thus it is not amenable to parametric search, which is the standard technique for turning the decision procedure into an efficient optimization procedure for solving the RSP problem. In this work we overcome these issues, for the special case of ball-intersection graphs in $\mathbb{R}^3$ (and in higher dimensions), and obtain efficient algorithms for solving the problem, albeit they are (not surprisingly) less efficient than in the planar case.

Related work. Breadth-First Search (BFS) is recognized as one of the fundamental graph algorithms. On general graphs $G = (V, E)$, its running time is $O(|V|+|E|)$, which is inefficient when $|E|$ is large. Its importance has led authors to seek families of graphs in which BFS can be implemented more efficiently. In the geometric context, one such example is the family of disk graphs in the plane. The vertices of a disk graph represent disks in the plane and there exists an edge between two vertices if and only if the corresponding disks intersect. Notice that the number of edges in a disk graph can be quadratic in n, the number of disks. Nevertheless, for unit-disk graphs (where all the disks are congruent), Cabello and Jejčič [8] presented an $O(n \log n)$ implementation of BFS, and subsequently Chan and Skrepetos [10] presented an alternative $O(n)$ implementation (after pre-sorting the points by their x- and y-coordinates). For arbitrary disks, Klost [14] described an $O(n \log^2 n)$ implementation of BFS (see also [11, 12]).

The proximity graph of a set S of n segments in the plane, for a real parameter $r \geq 0$, is $G_r(S) := (S, E)$, where $E = \{(e_1, e_2) \mid \text{dist}(e_1, e_2) \leq r\}$ and $\text{dist}(e_1, e_2)$ is the Euclidean distance between e_1 and e_2 (which is 0 if they intersect). Agarwal et al. [5] devised an $O^*(n^{4/3})$ implementation of BFS in $G_r(S)$. (As in the abstract, the $O^*(\cdot)$ notation hides subpolynomial factors.) For the special case where the segments in S are pairwise disjoint, Agarwal et al. [4] have later provided an $O(n \log^2 n)$ implementation.

The bottleneck path problem (also known as the minimax path problem) and its complementary problem, i.e., the widest path problem (or the maximum capacity problem), are well-known problems in graph theory. In suitable contexts, the bottleneck path problem is strongly related to the RSP problem: Given s and t, if π is a path from s to t of at most λ edges with the minimum bottleneck r^*, then r^* is the solution to the RSP problem with s, t, λ as input, and vice versa.

Abu-Affash et al. [1] studied the problem of placing at most k Steiner points to minimize the bottleneck of a path between two designated points in the plane, for a given integer parameter $k \geq 0$, and developed an $O(n \log^2 n)$-time algorithm for the problem.

The RSP problem for unit disks (as mentioned above) was studied by Wang and Zhao [16], who proposed an $O^*(n^{5/4})$-time solution. An improved solution with running time $O^*(n^{6/5})$, using the shrink-and-bifurcate technique (briefly reviewed later; see [7]), was subsequently presented by Kaplan et al. [11]. A very recent improvement of this technique by Chan and Huang [9] yields an implementation with running time $O^*(n^{8/7})$, which can be further improved, for the case of unit-disk graphs, to $O^*(n^{9/8})$ time. For arbitrary disks, Kaplan et al. [11] obtained a solution with randomized expected time $O^*(n^{5/4})$, which can be improved to randomized expected time $O^*(n^{6/5})$, using Chan and Huang's technique [9].

The RSP problem was also studied for other objects, including wedges of some fixed angle, which are viewed as directional antennas, and unit-height towers placed on a 1.5-dimensional terrain. The former version was solved in $O^*(n^{4/3})$ time by Agarwal et al. [5], and the latter version was studied in Katz et al. [13] (see also [9]).

Our results. Our main contributions are efficient algorithms for the BFS and RSP problems in ball-intersection graphs in three and higher dimensions. We first consider the problem of efficient implementation of BFS, as a decision procedure for guiding the search for the optimum r^* in the RSP problem (and also an interesting problem in itself). In three dimensions we can perform BFS on $G_0(\mathcal{B})$, for a set $\mathcal{B}$ of n (congruent or arbitrary) balls in $\mathbb{R}^3$, in $O^*(n^{4/3})$ time, and in d dimensions in $O^*(n^{2e/(e+1)})$ time, where $e = \lfloor d/2 \rfloor + 1$. As these algorithms are not parallelizable (at least we do not know how to run them in small parallel depth),[1] we need to use a different approach. Such an approach, already mentioned above, known as the *shrink-and-bifurcate* technique, has originally been developed in Ben Avraham et al. [7] for computing a rather special case of Fréchet distances. But it has recently found applications to the RSP problem in the plane [11]. Very recently, as already mentioned, it has been dramatically improved by Chan and Huang [9].

Once a BFS procedure (serving as the decision procedure) is available, our technique follows closely the improved technique in [9], which however requires certain nontrivial modifications and enhancements to fit into higher-dimensional contexts. The technique of [9] consists of two parts, a general-purpose part, and an additional enhancement which improves the running time further for the special case of unit-disk graphs. We follow the general-purpose part for arbitrary balls, and both parts for congruent balls.

In three dimensions, for the special case of congruent balls (equivalently, of a set of n points in $\mathbb{R}^3$), we obtain an algorithm that runs in $O^*(n^{29/21})$ randomized expected time. For the case of balls of arbitrary radii, we obtain an algorithm that runs in $O^*(n^{56/39})$ randomized expected time. More generally, in $d \geq 3$ dimensions, the algorithm for arbitrary balls runs in

$$O^*\left(n^{\frac{2(d+1)(3e+1)}{(3d+4)(e+1)}}\right)$$

randomized expected time, and, for the special case of congruent balls, the algorithm runs in randomized expected time

$$O^*\left(n^{\frac{g}{2e/(e+1)+g}+\frac{2e}{e+1}}\right), \text{ where } g = \frac{2(d-e)}{(e+1)(3d+1)}.$$

[1] This is a standard issue in BFS on any graph.

2 BFS in ball-intersection graphs

Let $\mathcal{B}$ be a collection of n balls in $\mathbb{R}^3$, of arbitrary radii.[2] We denote a ball with center c and radius ρ as $B(c,\rho)$. Let $G_0(\mathcal{B})$ be the intersection graph of $\mathcal{B}$, i.e., the vertices of $G_0(\mathcal{B})$ are the balls of $\mathcal{B}$, and its edges consist of all the intersecting pairs of balls. The condition for two balls $B(c_1,\rho_1)$, $B(c_2,\rho_2)$ to intersect is $|c_1c_2| \leq \rho_1+\rho_2$.

Let B_s be a start ball in $\mathcal{B}$. Our goal is to run BFS on $G_0(\mathcal{B})$ starting from B_s. We follow the standard approach, of constructing the layers of the BFS in order, starting from layer L_0 that contains only B_s. Consider the step of passing from some layer L_i to the next layer L_{i+1}. Let $\mathcal{U}_i$ denote the set of all balls that the BFS has not yet reached up to this step. We then need to find, for each ball $B=B(c,\rho)$ in L_i, all the balls of $\mathcal{U}_i$ that it intersects. Each such ball is added to L_{i+1} and is immediately deleted from $\mathcal{U}_i$, to ensure that it is not detected again by other balls of L_i (or by balls in future layers). We continue in this way until no more balls of $\mathcal{U}_i$ intersect any ball in L_i. At this point L_{i+1} has been computed, and we go on to construct the next layer. (For our decision procedure, executed on $G_r(\mathcal{B})$, we stop when either the target ball B_t has been reached or layer L_λ has been constructed, whichever happens first. If B_t has been reached, we report success (i.e., $r^* \leq r$), otherwise we report failure (i.e., $r^* > r$).)

To implement this algorithm efficiently, we therefore need a dynamic data structure for storing $\mathcal{U}=\mathcal{U}_i$, the set of unreached balls, that supports (efficiently) the following two kinds of operations.

Intersection detection queries: Given a query ball B, detect whether it intersects any ball of $\mathcal{U}$, and, if so, report such a ball.

Deletions: Delete a ball B from $\mathcal{U}$.

An intersection query with a ball $B(c_0,\rho_0)$ can be rephrased as a range emptiness detection query. For this, we map each ball $B=B(c,\rho)=B((x_c,y_c,z_c),\rho)$ of $\mathcal{U}$ to the point $\hat{B}=(x_c,y_c,z_c,\rho)$ in $\mathbb{R}^4$, and map each ball $B_0=B(c_0,\rho_0)=B((x_{c_0},y_{c_0},z_{c_0}),\rho_0)$ of L_i to the range

$$\sigma_{B_0}=\{(x,y,z,\rho)\in\mathbb{R}^4 \mid (x-x_{c_0})^2+(y-y_{c_0})^2+(z-z_{c_0})^2 \leq (\rho+\rho_0)^2\}.$$

Then a ball B intersects B_0 iff $\hat{B}\in\sigma_{B_0}$.

Using a standard lifting transform, we further map each point $\hat{B}=(x_c,y_c,z_c,\rho)$ to the point $B^*=(x_c,y_c,z_c,\rho,x_c^2+y_c^2+z_c^2-\rho^2)$ in $\mathbb{R}^5$. Each range σ_{B_0} is lifted to the halfspace

$$h_{B_0} := \quad x_5 \leq 2x_{c_0}x+2y_{c_0}y+2z_{c_0}z+2\rho_0\rho+(\rho_0^2-x_{c_0}^2-y_{c_0}^2-z_{c_0}^2).$$

As is easily checked, $\hat{B}\in\sigma_{B_0}$ iff B^* lies in h_{B_0}.

In other words, in the transformed problem we have a set $\mathcal{B}^*=\{B^* \mid B\in\mathcal{B}\}$ of n points in $\mathbb{R}^5$, which we want to process into a dynamic data structure (under deletions) for answering halfspace range emptiness queries. The algorithm of Agarwal and Matoušek [6] provides such a structure. Specifically, for any given parameter $n\leq s\leq n^2$, the structure can be constructed to have size $O^*(s)$, an initial version of it can be constructed in $O^*(s)$ time, each query takes $O^*(n/s^{1/2})$ time, and each deletion takes amortized $O^*(s/n)$ time. Choosing $s=n^{4/3}$, we obtain a data structure of size $O^*(n^{4/3})$, so that each operation (query or deletion) on the structure takes $O^*(n^{1/3})$ time.

[2] The case of congruent balls does not lead to an improved BFS implementation, although the algorithm becomes considerably simpler; see a discussion at the end of the section.

The number of operations is $O(n)$, since each query either detects a new intersecting ball, which is promptly removed from $\mathcal{U}$, or is the last query performed with a ball, and each ball becomes a query only at the layer it belongs to. We conclude that the overall running time of the BFS is $O^*(n^{4/3})$, because the cost of maintaining and searching in the data structure dominates the cost of the other steps of the BFS. We thus obtain:

▶ **Theorem 1.** *BFS on the ball intersection graph of a set of n balls in $\mathbb{R}^3$ can be performed in $O^*(n^{4/3})$ time.*

The case of congruent balls. We first state the following theorem, whose results will be useful for handling congruent balls. Its proof is an easy consequence of the analysis in [6, 15], applied to the lifted points and halfspaces in $\mathbb{R}^5$, as above.

▶ **Theorem 2.** *(a) Given two sets P, Q of m and n points, respectively, in three dimensions, and a prameter $r > 0$, we can determine, for each $p \in P$ whether there exists a point $q \in Q$ such that $|pq| \leq r$, in overall time*

$$O^*\left(m^{2/3}n^{2/3} + m + n\right).$$

(b) Given two sets P, Q, as above, we can compute, for each $p \in P$, the nearest neighbor of p in Q, in overall time

$$O^*\left(m^{2/3}n^{2/3} + m + n\right).$$

(Note that (b) subsumes (a), but the algorithm in (a) is simpler, so we state it separately.)

When the balls of $\mathcal{B}$ are congruent, say all of radius r, we do not need the above dynamic (and rather involved) structure. Instead, we follow the approach in the planar setup of unit-disk graphs [8, 10]. That is, we form a grid of cell size $r/2$, and distribute the ball centers among its cells. When we perform the step of passing from L_i to L_{i+1}, we process each *active* grid cell, i.e., a cell that contains centers of balls in L_i. For each such cell τ, all unreached balls with centers in τ are immediately placed in L_{i+1}. Then, for each cell τ' in a suitable punctured constant-size neighborhood of τ (only such cells, and τ itself, can contain centers of balls in $\mathcal{U}_i$ that can be reached, in one step, from balls with centers in τ), we take the set $\mathcal{U}_i(\tau')$ of unreached balls with centers in τ' and test each such ball B whether its center lies within distance r from the center of some ball in $L_i(\tau)$, the set of balls of L_i with centers in τ. Theorem 2(a) implies that this reverse search can be implemented in time

$$O^*\left(|\mathcal{U}_i(\tau')|^{2/3}|L_i(\tau)|^{2/3} + |\mathcal{U}_i(\tau')| + |L_i(\tau)|\right).$$

The remaining straightforward implementation details, and the analysis of correctness and performance of the structure, as given in [8, 10, 11], carry over to the three-dimensional case, and imply that the overall cost of this implementation of the BFS is $O^*(n^{4/3})$ time.

3 Reverse shortest paths in unit-ball graphs in three dimensions

We can now solve the corresponding optimization problem, which is the *reverse shortest path (RSP) problem*, or, as it is also called, the *bounded-hop bottleneck path problem*, for unit-ball graphs in $\mathbb{R}^3$: Given a set P of n points in $\mathbb{R}^3$, two designated points $s, t \in P$, and a parameter $0 < \lambda < n$, find the minimum value r^* of r for which the associated graph $G_{r^*}(P)$ contains a path from s to t of at most λ edges.

We can solve the RSP problem using the aforementioned shrink-and-bifurcate technique [7, 9, 11]. We begin with a brief overview of the technique. It applies in more general setups, but let us restrict it to our setup of unit-ball graphs (and later also to intersection graphs of arbitrary balls, although the description below assumes that the balls are congruent). The shrinking stage constructs an interval I that contains r^* and at most L other critical values of r, each of which is the distance between two centers, where L is a prespecified parameter. It does so, in the improved approach of [9], by running a distance selection algorithm on various pairs of random samples of $\mathcal{B}$ (that is, on the sets of centers of the balls in the samples).

The distance selection algorithm, on which the shrinking procedure is based, takes, ignoring the shrinking aspect, $O^*(n^{3/2})$ time. More generally, it takes $O^*(m^{3/4}n^{3/4} + m + n)$ time for selecting distances between two sets of m and n points, respectively, which is the setup that arises when applying the technique of [9]. Using (a suitable adaptation to three dimensions of) the shrinking mechanism of Chan and Huang [9], the construction of the desired interval I can be performed in expected $O^*(n^{3/2}/L^{3/4})$ time.

This is followed by a bifurcation stage, in which we simulate the decision procedure (in our setup, the BFS procedure of the preceding section). When we reach a comparison, we resolve it right away if the critical value r that it induces lies outside I. If however r lies in I, we bifurcate, exploring both outcomes $r^* \le r$ and $r^* > r$. When we have either accumulated enough unresolved comparisons, or moved forward at least some number of steps in the simulation, along each path in the bifurcation tree, we stop and resolve all comparisons using binary search, guided by the (unsimulated) decision procedure. This shrinks I further, and we start a new phase of bifurcating simulation. As shown in [7, 11], with a suitable choice of parameters, the overall cost of the bifurcation is $O^*(L^{1/2}D(n))$, where $D(n) = O^*(n^{4/3})$ is the cost of the decision procedure (i.e., of the BFS).

Hence the overall cost of the RSP algorithm is $O^*(n^{3/2}/L^{3/4} + L^{1/2}n^{4/3})$. We choose L to balance these terms, that is, $L = n^{2/15}$, and then the running time is $O^*(n^{7/5})$. That is, we have our initial, albeit weaker, result:

▶ **Theorem 3.** *For a set P of n points in $\mathbb{R}^3$, two designated points $s, t \in P$, and an integer parameter $0 < \lambda < n$, we can find the minimum radius r^* for which the associated graph $G_{r^*}(P)$ contains a path from s to t of at most λ edges, in $O^*(n^{7/5})$ time.*

3.1 An improved solution

To obtain a faster algorithm, we follow the high-level approach of the second improvement in [9], with many nontrivial adjustments that cater to the three-dimensional nature of the problem. We follow the same setup as in the second implementation of the BFS algorithm. However, for technical reasons that will become clear later on, we first perform the following steps, having to do with the cell size of the grid. We first note that the precise size of the cells (which we have set to $r/2$ in the BFS algorithm) is not very important, as long as it is smaller than $r/\sqrt{3}$ (but not much smaller), so that the propagation within a cell is immediate. We can therefore proceed as follows. Note that, by the nature of the RSP problem, r^* must be between $|st|/2$ and $|st|/(2n)$ (or, more precisely, $|st|/(2\lambda)$). We therefore run binary search through this interval, using any of the BFS decision procedures of Section 2 to guide the search, and in $O(\log n)$ steps we locate r^* within a range $[r_1, (1+\varepsilon)r_1]$, for some suitable

small $\varepsilon > 0$. We then set r to be[3] $r_1/2$, and keep this size throughout the simulation. See below how this is used in the algorithm. We emphasize that this step precedes the actual simulation, as well as the step of distributing the points of P in the grid cells.

We introduce a new threshold parameter $\Delta \ll n$, whose concrete value will be set later, and classify each nonempty grid cell as either *heavy*, if it contains at least Δ points of P, or *light*, otherwise. The number of heavy cells is at most $k := n/\Delta$. We now compress each heavy cell τ into a single (symbolic) "heavy" point p_τ, and obtain a new compressed version of $G_r(P)$, which we denote as $H_r(\hat{P})$, where $\hat{P}$ consists of all the points in the light cells and of all the representative heavy points p_τ. The distance between any pair p, q of points in light cells remains the same, namely $|pq|$. The distance between a heavy point p_τ and a light point q (a point in a light cell) is $\mathrm{dist}(q, P_\tau) = \min_{p \in \tau} |qp|$. Finally, the distance between two heavy points p_τ, $p_{\tau'}$ is the distance determined by the bichromatic closest pair (BCP) in $P_\tau \times P_{\tau'}$.

Running BFS on $H_r(\hat{P})$ is performed similarly to the previous implementation. A single propagation step, at some iteration i, involving two adjacent cells τ, τ', is executed depending on whether one of τ, τ', or both, are heavy. If both cells are light, we proceed as before, incurring the cost

$$O^*\left(|U_i(\tau')|^{2/3}|L_i(\tau)|^{2/3} + |U_i(\tau')| + |L_i(\tau)|\right); \tag{1}$$

see Theorem 2(a). When τ is light and τ' is heavy, $U_i(\tau')$ (if nonempty) is the singleton representative point $p_{\tau'}$, and by running the all-nearest-neighbor procedure (Theorem 2(b)) on $L_i(\tau)$ and $P_{\tau'}$, we can determine whether this step should place $p_{\tau'}$ in L_{i+1}. The cost is then

$$O^*\left(|P_{\tau'}|^{2/3}|L_i(\tau)|^{2/3} + |P_{\tau'}| + |L_i(\tau)|\right). \tag{2}$$

A similar procedure is applied when τ is heavy and τ' is light. In this case each point of $U_i(\tau')$ searches for a near neighbor in P_τ, and the cost is

$$O^*\left(|U_i(\tau')|^{2/3}|P_\tau|^{2/3} + |U_i(\tau')| + |P_\tau|\right). \tag{3}$$

Finally, when both cells are heavy we need to compute the distance of the BCP in $P_\tau \times P_{\tau'}$. This step is handled differently because its naïve implementation is too expensive, because it does not exploit the smaller parameter Δ; see shortly below.

We sum the bounds in (1), (2) and (3), over all iterations i and pairs τ, τ' of adjacent cells that are active at the corresponding iteration, when both τ and τ' are light (eq. (1)), when τ is light and τ' is heavy (3), and when τ is heavy and τ' is light (2). Note that in each of the bounds (1)–(3), at least one of the sets ($L_i(\tau)$ and / or $U_i(\tau')$) is from a light cell, so its size is at most Δ.

Applying Hölder's inequality, one can show that the sums of the bounds in (1), (2) and (3) are all $O^*(n\Delta^{1/3})$. For example, summing the bound in (1) over all iterations i and active pairs τ, τ' of neighboring cells, and using the facts that (i) $|L_i(\tau)| \leq \Delta$, and (ii) each cell is active, or a neighbor of an active cell, at only $O(1)$ iterations, in each of these bounds, we get

[3] Note that this choice might force us to change the definition of the neighborhood of a cell to potentially contain more cells, but still only a constant number thereof.

$$O^*\left(\sum_i \sum_{\tau,\tau'} \left(|U_i(\tau')|^{2/3}|L_i(\tau)|^{2/3} + |U_i(\tau')| + |L_i(\tau)|\right)\right)$$
$$= O^*\left(\Delta^{1/3} \sum_i \sum_{\tau,\tau'} |U_i(\tau')|^{2/3}|L_i(\tau)|^{1/3}\right) + O^*(n)$$
$$= O^*\left(\Delta^{1/3} \left(\sum_i \sum_{\tau,\tau'} |U_i(\tau')|\right)^{2/3} \left(\sum_i \sum_{\tau,\tau'} |L_i(\tau)|\right)^{1/3}\right) + O^*(n) = O^*(n\Delta^{1/3}),$$

and similarly for the sums of the bounds in (2) and (3).

It remains to handle the case where both τ and τ' are heavy. Here we need to compute the distance of the BCP in $P_\tau \times P_{\tau'}$, over all adjacent heavy pairs (i.e., pairs where both τ and τ' are heavy). As above, this can be done in a total of $O^*(n^{4/3})$ time. Since it does not depend on Δ, this cost turns out to be too expensive to pay during the simulation of the procedure. To bypass this issue, we note that, once we have fixed the cell-size $r_1/2$ of the grid (as we did, ahead of the simulation), this step is independent of r or r^*, and can too be performed ahead of the simulation. We thus compute the BCPs for all pairs of adjacent heavy cells, once and for all. The overall cost, of both the binary search to fix the cell-size of the grid and the BCP computations, is easily seen to be $O^*(n^{4/3})$. In addition, after having computed all the heavy BCP distances, we run an additional binary search through them, to ensure that the interval with which we start the shrink-and-bifurcate procedure does not contain any heavy BCP distance in its interior.

All these considerations imply that we can run BFS on the compressed graph $H_r(\hat{P})$ in time $O^*(n\Delta^{1/3})$, after an initial phase that takes $O^*(n^{4/3})$ time, which we will execute only once, outside (and preceding) the actual simulation during the search for r^*.

We now run this simulation of the BFS on $H_{r^*}(\hat{P})$, using the shrink-and-bifurcate paradigm, as in [9] and as briefly reviewed above. As before, the shrinking procedure is based on distance selection in $\mathbb{R}^3$, but we apply it separately to each pair of adjacent grid cells, at least one of which is light. (This is justified as in [9], because r^* must arise as a distance within such a pair, unless it is the BCP distance of some heavy pair, which will have been detected during the preliminary stage.) Arguing as in [9], and adapting the setup to $\mathbb{R}^3$, the cost of the shrinking stage is $O^*(n\Delta^{1/2}/L^{3/4})$.

The subsequent bifurcation procedure runs, as before, in time $O^*(L^{1/2}D(n))$, where $D(n)$ is the cost of the decision procedure. Ignoring the cost of the preliminary part, which stays outside the simulation, we have $D(n) = O^*(n\Delta^{1/3})$. Hence the overall cost of the shrink-and-bifurcate procedure, adding back the preliminary cost as a one-time cost, is

$$O^*(n\Delta^{1/2}/L^{3/4} + L^{1/2}n\Delta^{1/3} + n^{4/3}). \tag{4}$$

Of course, so far we have only simulated the BFS on the compressed graph $H_{r^*}(\hat{P})$, so the output value r^* is still not the correct value. Denoting it by r_1^*, it is the smallest value of r for which $H_r(\hat{P})$ has a path of at most λ edges between s and t. As argued in [9], denoting by $d_H(s,p)$ the graph distance between s and a point p in $H_{r_1^*}(\hat{P})$, and by $d_G(s,p)$ the graph distance between s and p in $G_{r^*}(P)$, we have, for each p,

$$d_G(s,p) - k \le d_H(s,p) \le d_G(s,p), \tag{5}$$

where $k = n/\Delta$. With some care. this also covers the cases where s and / or t lie in heavy cells.

Recovering r^*

As in [9], to find the true value r^*, we use dynamic programming, following the high-level approach of [9]. For the sake of completeness, we repeat here some details of this technique, adapted to our setup. We first comment that it is easy to solve the RSP problem using dynamic programming, without using the preceding machinery at all. To do so, let $D_i(p)$, for $p \in P$, denote the smallest bottleneck value of a path from s to p consisting of at most i edges. Initially, $D_0(s) = 0$ and $D_0(p) = +\infty$ for any $p \neq s$. Then we have, for $i \geq 1$ and $p \in P$,

$$D_i(p) = \min_{q \in P} \max \{D_{i-1}(q),\ |qp|\}\,. \tag{6}$$

If $D_\lambda(t) < +\infty$ then $D_\lambda(t) = r^*$, which follows easily from (6). The problem with this approach is that the number of times the step (6) is applied is $\Theta(n^2)$ (it is actually $O(n\lambda)$, for the number of points n and the number of iterations λ to reach, or not to reach, t). Moreover, without any efficient implementation of the steps (6), the overall cost can be $\Theta(n^3)$.

In our setup, though, we can reduce the number of dynamic programming steps to $O(nk)$. Indeed, write $P_i = \{p \in P \mid i - k \leq d_H(p) \leq i\}$. As each point participates in $k+1$ sets P_i, we have $\sum_i |P_i| = O(nk)$.

As in [9], we exploit this property by replacing the function D by another function $\hat{D}$, defined by the following modified iterative process. For each $p \in P_i$ put

$$\hat{D}_i(p) = \min_{q \in P_{i-1}} \max \left\{\hat{D}_{i-1}(q),\ |qp|\right\}, \tag{7}$$

and $\hat{D}_0(s) = 0$ and $\hat{D}_0(p) = +\infty$ for all $p \neq s$. As shown in [9], we also have $\hat{D}_\lambda(t) = r^*$. (This argument is fully general and discrete, and does not depend on any geometric feature of the problem, except for the inequalities (5). In particular, it does not depend on the dimension.)

The number of steps in this modified dynamic programming process is thus $O(nk)$. To make these steps (relatively) efficient, we follow a variant of the technique in [9, Lemma 8], adapted to three dimensions. Briefly, the input is a set Q of n points in $\mathbb{R}^3$, so that each point $q \in Q$ carries a positive weight w_q (the weight w_q will be set to $\hat{D}_{i-1}(q)$ at the i-th iteration), and we have a set P of m queries, where each query point p seeks the point $q \in Q$ that minimizes $\max\{w_q, |qp|\}$. The algorithm in [9, Lemma 8] is based on Voronoi diagrams, which works well in the plane, but is not efficient in three dimensions. We use the following modified approach. As in [9], we store the points of Q at the leaves of a balanced binary tree T, sorted in increasing order of their weights. Each node v of T processes the set Q_v of those points of Q that are stored at the subtree rooted at v. Each query point p follows a path in T. When p reaches a node v, with a left child v_L and a right child v_R, it finds its (standard) nearest neighbor q_L in Q_{v_L}. If $|pq_L| \leq w_{q_L}$ we recurse in v_L, and if $|pq_L| > w_{q_L}$ we recurse in v_R. See [9] for details and justification of this rule.

Instead of using Voronoi diagrams, as in [9], we use the algorithm in Theorem 2(b). To do so, we run all the queries in parallel, proceeding level by level. When we reach a node v, we have available the set P_v of all the queries whose paths through T reach v. Using the algorithm in Theorem 2(b), we find, for each $p \in P_v$, its nearest neighbor in Q_{v_L}, in overall time

$$O^*\left(|P_v|^{2/3}|Q_{v_L}|^{2/3} + |P_v| + |Q_{v_L}|\right).$$

This allows us, by our search rule, to assign each $p \in P_v$ either to v_L or to v_R, as appropriate. Carrying out this process at each node of the current level, we obtain, for each node w of the next level, the set P_w of query points that want to access w (actually, w_L). We continue in this recursive manner until we reach the leaves. The cost at a leaf v is $O(1 + |P_v|)$. Summing over all nodes of T, bearing in mind that

$$\sum_v |P_v| = O(m \log n) \qquad \text{and} \qquad \sum_v |Q_{v_L}| = O(n \log n),$$

the overall cost is easily seen to be $O^*\left(|P|^{2/3}|Q|^{2/3} + |P| + |Q|\right)$.

We now apply this procedure at each iteration i of the dynamic programming, where the input set is P_{i-1}, the query set is P_i, and the weight of each input point q is $\hat{D}_{i-1}(q)$. Clearly, the output of this procedure, with this input, implements the iterative process in (7). The cost of iteration i is therefore

$$O^*\left(|P_i|^{2/3}|P_{i-1}|^{2/3} + |P_i| + |P_{i-1}|\right).$$

Since each $|P_i|$ is at most n, and $\sum_i |P_i| = O(nk)$, the overall cost of the dynamic program is (using Hölder's inequality)

$$\begin{aligned}
&O^*\left(\sum_i \left(|P_i|^{2/3}|P_{i-1}|^{2/3} + |P_i| + |P_{i-1}|\right)\right) \\
&= n^{1/3} O^*\left(\sum_i \left(|P_i|^{1/3}|P_{i-1}|^{2/3}\right)\right) + O^*\left(\sum_i (|P_i| + |P_{i-1}|)\right) \\
&= O^*\left(n^{1/3}(nk)^{1/3}(nk)^{2/3} + nk\right) = O^*(n^{4/3}k) = O^*\left(\frac{n^{7/3}}{\Delta}\right). \qquad (8)
\end{aligned}$$

Altogether, the cost of the full algorithm is therefore

$$O^*\left(\frac{n\Delta^{1/2}}{L^{3/4}} + L^{1/2}n\Delta^{1/3} + n^{4/3} + \frac{n^{7/3}}{\Delta}\right) = O^*\left(\frac{n\Delta^{1/2}}{L^{3/4}} + L^{1/2}n\Delta^{1/3} + \frac{n^{7/3}}{\Delta}\right). \qquad (9)$$

We first balance the first two terms, choosing $L^{5/4} = \Delta^{1/6}$, or $L = \Delta^{2/15}$, and then the bound becomes

$$O^*\left(n\Delta^{2/5} + \frac{n^{7/3}}{\Delta}\right).$$

We now choose Δ to balance these terms, that is $\Delta = n^{20/21}$, and the cost of the algorithm is therefore $O^*(n^{29/21})$. That is, we have

▶ **Theorem 4.** *Let P be a set of n points in $\mathbb{R}^3$, let s and t be two designated points, and let $0 < \lambda < n$ be a given parameter. Then we can find the smallest value r^* for which $G_{r^*}(P)$ contains a path between s and t that consists of at most λ edges, in $O^*(n^{29/21}) = O^*(n^{1.381})$ time.*

3.2 Higher dimensions

The algorithm given above can be generalized to points in $\mathbb{R}^d$, for any dimension $d \geq 4$. The geometric components in the algorithm are (i) constructing the grid, (ii) distance selection and the corresponding interval shrinking procedure, (iii) an efficient algorithm for the all-near-neighbor problem, (iv) an efficient algorithm for the all-nearest-neighbor problem, and, as a special case, (v) an efficient algorithm for the BCP problem.

The grid construction can trivially be extended to any fixed dimension. Distance selection in $\mathbb{R}^d$ takes $O^*(n^{2d/(d+1)})$ time (see, e.g., [3, Appendix]). The bipartite version, where we want to select distances between two given sets of m and n points, respectively, takes $O^*(m^{d/(d+1)}n^{d/(d+1)} + m + n)$ time, and the corresponding interval shrinking procedure, implemented using the technique of [9], takes $O^*(m^{d/(d+1)}n^{d/(d+1)}/L^{d/(d+1)})$ expected time, as long as m and n do not deviate significantly from one another. Agarwal and Matoušek [6] present an algorithm that solves problems (iii)–(v) for a pair of sets of respective sizes m and n, in $\mathbb{R}^d$ in time $O^*\left(m^{e/(e+1)}n^{e/(e+1)} + m + n\right)$, where $e = \lfloor d/2 \rfloor + 1$. (Recall that these problems are mapped, via the lifting transform given in Section 2, to dynamic halfspace range emptiness queries in $d+2$ dimensions, which explains the value of e; see [6].)

Substituting these components into the algorithm, with parameters L and Δ, the running time becomes

$$O^*\left(\frac{n\Delta^{(d-1)/(d+1)}}{L^{d/(d+1)}} + L^{1/2}n\Delta^{(e-1)/(e+1)} + n^{2e/(e+1)} + \frac{n^{(3e+1)/(e+1)}}{\Delta}\right),$$

where, as above, the third term is dominated by the fourth. We balance the terms in this bound, first by choosing L as a function of Δ, and then by choosing Δ as a function of n. As the exponents become rather messy, we simplify them somewhat by introducing another parameter $g = \frac{2(d-e)}{(e+1)(3d+1)}$. Straightforward, albeit rather tedious, calculations then yield

$$L = \Delta^{2g} \qquad \text{and} \qquad \Delta = n^{\frac{2e/(e+1)}{2e/(e+1)+g}},$$

and the bound then becomes

$$O^*\left(n^{\frac{g}{2e/(e+1)+g}+\frac{2e}{e+1}}\right).$$

As a sanity check, we verify that in $\mathbb{R}^3$, where $d = 3$, $e = 2$, and $g = 1/15$, the bound is indeed $O^*(n^{29/21})$. In conclusion, we have

▶ **Theorem 5.** *Let P be a set of n points in $\mathbb{R}^d$, for $d \geq 3$, let s and t be two designated points, and let $0 < \lambda < n$ be a given parameter. Then we can find the smallest value r^* for which $G_{r^*}(P)$ contains a path between s and t that consists of at most λ edges, in $O^*\left(n^{\frac{g}{2e/(e+1)+g}+\frac{2e}{e+1}}\right)$ randomized expected time, where $e = \lfloor d/2 \rfloor + 1$ and $g = \frac{2(d-e)}{(e+1)(3d+1)}$.*

As an additional example, in the case $d = 4$, with $e = 3$ and $g = 1/26$, the algorithm runs in $O^*(n^{61/40}) = O^*(n^{1.525})$ randomized expected time.

4 Reverse shortest paths in ball-intersection graphs in $\mathbb{R}^3$

As before, we use the shrink-and-bifurcate paradigm (see [7, 9, 11]). As we recall, the shrinking stage receives a number $L \ll n$ as input, and constructs an interval I that contains r^* and at most L other critical values of r. For arbitrary balls, the problem is represented in $\mathbb{R}^4$, where a ball with center c and radius ρ is mapped to the point (c, ρ). A value r is critical if two expanded balls $B_r(c_1, \rho_1) = B(c_1, \rho_1 + r)$ and $B_r(c_2, \rho_2) = B(c_2, \rho_2 + r)$ become externally tangent to each other, namely when

$$|c_1c_2| = (\rho_1 + r) + (\rho_2 + r), \qquad \text{or} \qquad r = \frac{1}{2}\left(|c_1c_2| - \rho_1 - \rho_2\right).$$

The improved shrinking procedure of [9] performs "distance selection" on various random samples from $\mathcal{B}$, where the selection has to find the k-th smallest critical value between pairs of balls in the cartesian product of the samples, for some given k. This selection process

is obtained in turn from a procedure that counts the number of critical values that are smaller than or equal to some given r_0. To do so, we rephrase the problem as an offline range searching problem, in which each ball $B = B(c_1, \rho_1)$ is mapped to the point (c_1, ρ_1) in $\mathbb{R}^4$, and also to the range[4]

$$\sigma_r(B) = \{(c, \rho) \in \mathbb{R}^4 \mid |cc_1| - \rho \leq 2r + \rho_1\}.$$

Each point has four degrees of freedom, and, for r fixed, so does each range. Using known results on semi-algebraic range searching (see [2] and [3, Appendix]), the cost of this procedure is $O^*(n^{8/5})$. In fact, running this "distance selection" procedure in a bipartite setting (which is the setting that arises when applying the technique of [9]), where we want to select critical values determined between pairs of balls in a pair of samples, consisting of m and n balls, respectively, the cost is $O^*(m^{4/5}n^{4/5} + m + n)$. Plugging this into the machinery of [9], the cost of the shrinking stage is $O^*(n^{8/5}/L^{4/5})$.

The cost of the bifurcation stage, as in all the previous applications, is $O^*(L^{1/2}D(n))$, where $D(n)$ is a bound on the running time of the decision procedure, which is our BFS algorithm on $G_r(\mathcal{B}) = G_0(\mathcal{B}_r)$, so $D(n) = O^*(n^{4/3})$. Hence the overall cost of the procedure is

$$O^*\left(\frac{n^{8/5}}{L^{4/5}} + L^{1/2}n^{4/3}\right).$$

We optimize this bound by choosing $L^{13/10} = n^{4/15}$, or $L = n^{8/39}$, and then the running time becomes $O^*(n^{56/39})$. That is, we have:

▶ **Theorem 6.** *The reverse shortest path problem on the ball-intersection graph of a set of n balls in $\mathbb{R}^3$ can be solved in $O^*(n^{56/39}) \approx O^*(n^{1.436})$ time.*

4.1 Higher dimensions

Here too, the machinery developed in Section 2 and the preceding part of Section 4 can be extended to any higher dimension d.

Breadth-First Search. We proceed as in the previous algorithm. The lifting transform used in the BFS implementation maps each ball $B_0 = B(c_0, \rho_0)$ to the point $B^* = (c_0, \rho_0, |c_0|^2 - \rho_0^2)$ in $\mathbb{R}^{d+2}$, and also to the halfspace

$$h_{B_0}: \qquad x_{d+2} \leq 2c_0 \cdot c + 2\rho_0\rho + \rho_0^2 - |c_0|^2$$

in $\mathbb{R}^{d+2}$. The dynamic halfspace range reporting structure of [6] yields, for any parameter $n \leq s \leq n^e$, where $e = \lfloor (d+2)/2 \rfloor = \lfloor d/2 \rfloor + 1$, a dynamic data structure of size $O^*(s)$, with initial construction cost $O^*(s)$, so that each query takes $O^*(n/s^{1/e})$ time, and each deletion takes $O^*(s/n)$ amortized time. Choosing $s = n^{2e/(e+1)}$, the size (and construction cost) becomes $O^*(s) = O^*(n^{2e/(e+1)})$, and each operation takes $O^*(n^{(e-1)/(e+1)})$ time, for a total of $O^*(n^{2e/(e+1)})$ time. Since this dominates the running time of the BFS, we obtain:

▶ **Theorem 7.** *BFS on the ball intersection graph of a set of n balls in $\mathbb{R}^d$ can be performed in $O^*(n^{2e/(e+1)})$ randomized expected time, where $e = \lfloor d/2 \rfloor + 1$.*

[4] Here the lifting transform to a halfspace in $\mathbb{R}^5$ does not seem to lead to a more efficient procedure. Technically, this is because the lifted problem to $\mathbb{R}^5$ now has to deal with halfspace range *counting*, whose performance is actually worse than the corresponding problem (involving semi-algebraic ranges instead of halfspaces) in $\mathbb{R}^4$.

Reverse Shortest Paths. Here the selection procedure, rephrased as an offline semi-algebraic range searching problem, involves points and ranges, each point and range with $d+1$ degrees of freedom. Hence the running time of this procedure, in the bipartite setting of sets of m and n (ball-representing) points, respectively, is (again, see [3, Appendix])

$$O^*\left(m^{(d+1)/(d+2)}n^{(d+1)/(d+2)} + m + n\right).$$

This implies, as in [9], that the cost of the interval shrinking procedure is

$$O^*\left(n^{2(d+1)/(d+2)}/L^{(d+1)/(d+2)}\right).$$

As in all its implementations, the bifurcation procedure runs in time

$$O^*\left(L^{1/2}D(n)\right) = O^*\left(L^{1/2}n^{2e/(e+1)}\right).$$

The overall running time is thus

$$O^*\left(n^{2(d+1)/(d+2)}/L^{(d+1)/(d+2)} + L^{1/2}n^{2e/(e+1)}\right).$$

Optimizing the value of L, we choose

$$L^{\frac{1}{2}+\frac{d+1}{d+2}} = n^{\frac{2(d+1)}{d+2}-\frac{2e}{e+1}}, \qquad \text{or} \qquad L = n^{\frac{4(d-e+1)}{(3d+4)(e+1)}},$$

and the overall running time of the algorithm is

$$O^*\left(n^{\frac{2(d+1)(3e+1)}{(3d+4)(e+1)}}\right).$$

(We verify that, for $d = 3$ and $e = 2$, this bound is indeed $O^*(n^{56/39})$.) That is, we have:

▶ **Theorem 8.** *The reverse shortest path problem on the ball intersection graph of a set of n balls in $\mathbb{R}^d$ can be solved in $O^*\left(n^{\frac{2(d+1)(3e+1)}{(3d+4)(e+1)}}\right)$ time, where $e = \lfloor d/2 \rfloor + 1$.*

(We remind the reader that the second improvement in [9] is not applicable here, when the balls do not have the same radius.)

For example, in $d = 4$ dimensions, with $e = 3$, the algorithm runs in $O^*(n^{25/16}) = O^*(n^{1.5625})$ time.

5 Further extensions

Other measures of expansion. The RSP technique can easily be applied to other well-behaved measures of expansion of the balls. For example, consider, in three dimensions, the case where, for $r \geq 1$, each ball $B(c, \rho)$ is expanded to the ball $B(c, r\rho)$ (all radii are multiplied by r). Now r is a critical value iff a pair $B(c_1, r\rho_1)$, $B(c_2, r\rho_2)$ of balls become externally tangent to each other, that is,

$$|c_1c_2| = r(\rho_1 + \rho_2), \qquad \text{or} \qquad r = \frac{|c_1c_2|}{\rho_1 + \rho_2}$$

Hence, in the selection procedure, we map each ball $B_0 = B(c_0, \rho_0)$ to the range

$$\sigma_{B_0} = \{(c, \rho) \in \mathbb{R}^4 \mid |cc_0| = r(\rho + \rho_0)\}.$$

Since these ranges are also semi-algebraic regions of constant complexity with four degrees of freedom, the same previously used offline semi-algebraic range searching machinery (on a different kind of ranges) can be applied here too, with the same asymptotic running time bound. The remainder of the procedure is carried out as before. Any other expansion rule can also be handled in this manner, as long as the corresponding ranges σ_{B_0} are semi-algebraic of constant complexity (with four degrees of freedom).

The same observations hold in any larger dimension d.

In conclusion, Theorems 4, 5, 6 and 8 continue to hold for any well-behaved measure of expansion.

References

1 A. Karim Abu-Affash, Paz Carmi, Matthew J. Katz, and Michael Segal. The Euclidean bottleneck Steiner path problem and other applications of (α, β)-pair decomposition. *Discret. Comput. Geom.*, 51(1):1–23, 2014. doi:10.1007/S00454-013-9550-9.

2 Pankaj K. Agarwal. Simplex range searching and its variants: A review. In *Journey through Discrete Mathematics: A Tribute to Jiří Matoušek*, pages 1–30. Springer Verlag, Berlin-Heidelberg, 2017.

3 Pankaj K. Agarwal, Boris Aronov, Esther Ezra, Matthew J. Katz, and Micha Sharir. Intersection queries for flat semi-algebraic objects in three dimensions and related problems. *ACM Trans. Algorithms*, 21(3):25:1–25:59, 2025. doi:10.1145/3721290.

4 Pankaj K. Agarwal, Haim Kaplan, Matthew J. Katz, and Micha Sharir. Segment proximity graphs and nearest neighbor queries amid disjoint segments. In *32nd Annual European Symposium on Algorithms, ESA 2024*, volume 308 of *LIPIcs*, pages 7:1–7:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ESA.2024.7.

5 Pankaj K. Agarwal, Matthew J. Katz, and Micha Sharir. On reverse shortest paths in geometric proximity graphs. *Comput. Geom.*, 117:102053, 2024. doi:10.1016/J.COMGEO.2023.102053.

6 Pankaj K. Agarwal and Jirí Matousek. Dynamic half-space range reporting and its applications. *Algorithmica*, 13(4):325–345, 1995. doi:10.1007/BF01293483.

7 Rinat Ben Avraham, Omrit Filtser, Haim Kaplan, Matthew J. Katz, and Micha Sharir. The discrete and semicontinuous Fréchet distance with shortcuts via approximate distance counting and selection. *ACM Trans. Algorithms*, 11(4):29:1–29:29, 2015. doi:10.1145/2700222.

8 Sergio Cabello and Miha Jejcic. Shortest paths in intersection graphs of unit disks. *Comput. Geom.*, 48(4):360–367, 2015. doi:10.1016/J.COMGEO.2014.12.003.

9 Timothy M. Chan and Zhengcheng Huang. Faster algorithms for reverse shortest path in unit-disk graphs and related geometric optimization problems: Improving the shrink-and-bifurcate technique. In *41st International Symposium on Computational Geometry, SoCG 2025*, volume 332 of *LIPIcs*, pages 32:1–32:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPICS.SOCG.2025.32.

10 Timothy M. Chan and Dimitrios Skrepetos. All-pairs shortest paths in unit-disk graphs in slightly subquadratic time. In *27th International Symposium on Algorithms and Computation, ISAAC 2016*, volume 64 of *LIPIcs*, pages 24:1–24:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICS.ISAAC.2016.24.

11 Haim Kaplan, Matthew J. Katz, Rachel Saban, and Micha Sharir. The unweighted and weighted reverse shortest path problem for disk graphs. In *31st Annual European Symposium on Algorithms, ESA 2023*, volume 274 of *LIPIcs*, pages 67:1–67:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ESA.2023.67.

12 Haim Kaplan, Wolfgang Mulzer, Liam Roditty, Paul Seiferth, and Micha Sharir. Dynamic planar Voronoi diagrams for general distance functions and their algorithmic applications. *Discret. Comput. Geom.*, 64(3):838–904, 2020. doi:10.1007/S00454-020-00243-7.

13 Matthew J. Katz, Rachel Saban, and Micha Sharir. Near-linear algorithms for visibility graphs over a 1.5-dimensional terrain. In *32nd Annual European Symposium on Algorithms, ESA 2024*, volume 308 of *LIPIcs*, pages 77:1–77:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ESA.2024.77.

14 Katharina Klost. An algorithmic framework for the single source shortest path problem with applications to disk graphs. *Comput. Geom.*, 111:101979, 2023. doi:10.1016/J.COMGEO.2022.101979.

15 Jirí Matousek. Reporting points in halfspaces. *Comput. Geom.*, 2:169–186, 1992. doi:10.1016/0925-7721(92)90006-E.

16 Haitao Wang and Yiming Zhao. Reverse shortest path problem for unit-disk graphs. *J. Comput. Geom.*, 14(1):14–47, 2023. doi:10.20382/JOCG.V14I1A2.

Realizing Metric Spaces with Convex Obstacles

Sándor Kisfaludi-Bak
Aalto University, Finland

Leonidas Theocharous
University of Ottawa, Canada

Abstract

The presence of obstacles has a significant impact on distance computation, motion-planning, and visibility. These problems have been studied extensively in the planar setting, while our understanding of these problems in 3- and higher-dimensional spaces is still rudimentary. In this paper, we study the impact of different types of obstacles on the induced geodesic metric in 3-dimensional Euclidean space. We say that a finite metric space (X, dist_X) is *approximately realizable* by a collection $\mathcal{T}$ of obstacles in $\mathbb{R}^3$ if for any $\varepsilon > 0$ it can be embedded into $(\mathbb{R}^3 \setminus \bigcup_{T \in \mathcal{T}} T, \mathrm{dist}_{\mathcal{T}})$ with worst-case multiplicative distortion $1 + \varepsilon$, where $\mathrm{dist}_{\mathcal{T}}$ denotes the geodesic distance in the free space induced by $\mathcal{T}$. We focus on three key geometric properties of obstacles –convexity, disjointness, and fatness– and examine how dropping each one of them affects the existence of such embeddings.

Our main result concerns dropping the fatness property: we demonstrate that any finite metric space is realizable with $1 + \varepsilon$ worst-case multiplicative distortion using a collection of convex and pairwise disjoint obstacles in $\mathbb{R}^3$, even if the obstacles are congruent and equilateral triangles. Based on the same construction, we can also show that if we require fatness but drop any of the other two properties instead, then we can still approximately realize any finite metric space.

Our results have important implications on the approximability of TSP WITH OBSTACLES, a natural variant of TSP introduced recently by Alkema *et al.* (ESA 2022). Specifically, we use the recent results of Banerjee *et al.* on TSP in doubling spaces (FOCS 2024) and of Chew *et al.* on distances among obstacles (Inf. Process. Lett. 2002) to show that TSP WITH OBSTACLES admits a PTAS if the obstacles are convex, fat, and pairwise disjoint. If any of these three properties is dropped, then our results, combined with the APX-hardness of METRIC TSP, demonstrate that TSP WITH OBSTACLES is APX-hard.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases traveling salesman, geodesic distance

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.46

Related Version *Full Version*: https://arxiv.org/abs/2509.14709 [26]

Funding *Sándor Kisfaludi-Bak*: This work was supported by the Research Council of Finland, Grant 363444.

1 Introduction

Understanding navigation [3, 10, 20, 36, 37], distance [5, 23, 28, 38, 40, 41] and visibility [12, 16, 27, 29, 33, 35] problems among obstacles has been one of the well-motivated research directions in computational geometry. Most of the work has focused on the planar setting, where the metric could be the geodesic distance inside a simple polygon, or more generally in a polygon with holes in the Euclidean plane. In the plane one can benefit from the fact that paths will often cross, whereas in higher dimensions paths will typically stay disjoint. The additional degree of freedom in 3- and higher-dimensional space makes the fundamental distance and navigation problems among obstacles harder, and they have not been explored as thoroughly.

Fortunately, we can still get a useful metric structure for certain obstacles in d-dimensional Euclidean space (henceforth denoted by $\mathbb{R}^d$). We say that a set $\mathcal{T}$ of obstacles is α-*fat* if for each obstacle object $T \in \mathcal{T}$ we have $\frac{r_{in}(T)}{r_{circum}(T)} \geqslant \alpha$, where $r_{in}(T)$ denotes the radius of the

© Sándor Kisfaludi-Bak and Leonidas Theocharous;
licensed under Creative Commons License CC-BY 4.0

maximum inscribed ball of T, and $r_{circum}(T)$ denotes the radius of the minimum enclosing ball of T. We will think of α as a universal constant and talk about collections of fat objects. For a set $\mathcal{T}$ of obstacles let $\mathcal{F}_{\mathcal{T}} = \mathbb{R}^d \setminus \bigcup_{T \in \mathcal{T}} T$ denote the *free space* outside the obstacles. We denote by $\mathrm{dist}_{\mathcal{T}}(u, v)$ the geodesic distance between $u, v \in \mathcal{F}_{\mathcal{T}}$, which is the infimum of the length of piecewise linear curves in $\mathcal{F}_{\mathcal{T}}$ containing u and v. Chew et al. [11] established that for any set of convex, disjoint and fat obstacles in $\mathbb{R}^d$ one can navigate around the obstacles with constant overhead compared to the Euclidean distance. Their theorem implies that $(\mathcal{F}_{\mathcal{T}}, \mathrm{dist}_{\mathcal{T}})$ has bounded *doubling dimension*, that is, for any $r > 0$ and any $x \in \mathcal{F}_{\mathcal{T}}$ the geodesic ball of center x and radius r can be covered by at most 2^δ geodesic balls of radius $r/2$, for some constant δ. The minimum number δ satisfying the above condition is the doubling dimension. In Section 2 we establish an explicit bound on the doubling dimension among pairwise disjoint convex α-fat obstacles in $\mathbb{R}^d$. Metrics of bounded doubling dimension are well-studied generalizations of Euclidean spaces, and retain many useful properties of $\mathbb{R}^d$.

A natural question is whether the conditions of fatness, convexity, and pairwise disjointness are all needed for the doubling dimension bound. More generally, one wonders what finite metric spaces can be realized if the condition of fatness, disjointness or convexity is dropped. We say that a metric space (X, dist_X), is *realizable* with a set $\mathcal{T}$ of obstacles in $\mathbb{R}^d$ if one can embed (X, dist_X) isometrically into $(\mathcal{F}_{\mathcal{T}}, \mathrm{dist}_{\mathcal{T}})$. An *approximate realization* allows for an embedding with a worst-case multiplicative distortion of $(1 + \varepsilon)$.

The starting point of our investigation is the following question. Let $\mathfrak{P}$ denote the disjunction of some subset of properties among (i) convex (ii) pairwise disjoint (iii) fat.

▶ **Question 1.** *Is it possible to (approximately) realize the finite metric space* (X, dist_X) *in* $\mathbb{R}^3$ *with obstacles of property* $\mathfrak{P}$*?*

It is also worth exploring how various types of obstacles impact the algorithms and complexity of some classic distance problems. The study of the shortest path problem among a set of polygonal obstacles $\mathcal{P}$ goes back at least 50 years, to the work of Wangdahl et al. [41] in 1974. Their approach to compute a shortest path between two points s, t (as well as many later approaches; see for instance [5, 38]) relied on running Dijkstra's algorithm on the visibility graph of the vertices of $\mathcal{P}$ together with s and t. Any such method has an $\Omega(n^2)$ lower bound (where n is the number of vertices in $\mathcal{P}$), due to the size of the visibility graph. To achieve subquadratic running times, later algorithms shifted to the continuous Dijkstra paradigm [23, 28]. Finally, Wang presented an optimal $O(n \log n)$-time, $O(n)$-space algorithm [40]. Data structures for two-point shortest path queries have also been studied and, recently, De Berg et al. presented an $O(n^{10+\varepsilon})$-space data structure with $O(\log n)$ query time [14]. The shortest path problem among obstacles can be generalized to account for movement through regions with different weights, a variation known as the *weighted region problem*. The general problem admits a $(1 + \varepsilon)$-approximation algorithm [30], while cases with restricted region shapes or weights have also been considered [9, 17]. While these are all positive results, computing shortest paths among 3-dimensional obstacles is a much harder problem. In fact, it is NP-hard [31]. On the other hand, the problem admits an FPTAS [13, 21].

It is also natural to consider classic geometric optimization problems in the presence of obstacles. Recently, Alkema et al. [4] introduced a natural generalization of Euclidean TSP called TSP WITH OBSTACLES. In TSP WITH OBSTACLES, we are given a set of n points to visit and a collection $\mathcal{T}$ of obstacles of total complexity m to avoid. Our goal is to find the shortest tour between a set of sites in the plane or some higher-dimensional space, while avoiding a given set of obstacles. In [4] the authors studied a variant of TSP WITH OBSTACLES called

TSP IN A SIMPLE POLYGON, where the sites lie in a simple polygon P, that is, the obstacle that the salesman has to avoid is the complement of P. They gave an exact algorithm for this problem, running in $2^{O(\sqrt{n}\log n)} + \text{poly}(n, m)$ time.

In this paper, we will demonstrate the algorithmic impact of various types of obstacles via studying approximation algorithms and APX-hardness for TSP WITH OBSTACLES. In particular, we consider the problem in $\mathbb{R}^3$ for different kinds of obstacles. Again, let $\mathfrak{P}$ denote the disjunction of some subset of properties among (i) convex (ii) pairwise disjoint (iii) fat.

▶ **Question 2.** *Is there a PTAS for* TSP WITH OBSTACLES *in* $\mathbb{R}^3$ *if the obstacles satisfy property* $\mathfrak{P}$*?*

1.1 Our contribution

Our main contribution is the following theorem, which shows that all finite metric spaces are approximately realizable with pairwise disjoint convex obstacles in $\mathbb{R}^3$.

▶ **Theorem 3.** *Let* $\varepsilon \in (0,1)$ *and let* (X, dist_X) *be a metric space of size* $|X| = n$ *and spread* $\Phi := \max_{a,b,c,d \in X}(\frac{\text{dist}_X(a,b)}{\text{dist}_X(c,d)})$*. Then there exists a collection* $\mathcal{T}$ *of* $O(n^{17}\Phi^5/\varepsilon^5)$ *pairwise disjoint congruent equilateral triangular obstacles and an injection* $f : X \to \mathbb{R}^3$ *such that*

$$\text{dist}_X(a,b) \leqslant \text{dist}_{\mathcal{T}}(f(a), f(b)) \leqslant (1+\varepsilon) \cdot \text{dist}_X(a,b).$$

for all $a, b \in X$*. The set of obstacles can be constructed in* $\text{poly}(n, \Phi, 1/\varepsilon)$ *time.*

Let $\mathfrak{P}$ denote the disjunction of some subset of properties among (i) convex (ii) pairwise disjoint (iii) fat. The theorem directly answers Question 1 if the property $\mathfrak{P}$ does not require that objects are fat. We can use some ideas from this construction to give a complete answer to Question 1 in case of approximate realizations: we can approximately realize any metric space with fat pairwise disjoint obstacles (that are non necessarily convex) and with convex fat obstacles that are not necessarily disjoint, see Theorem 15.

We note that our construction is quite generic and we get the same result with other convex shapes that are constant-fat in $\mathbb{R}^2$, such as with unit disks or squares instead of congruent equilateral triangles. Using Theorem 3 and Theorem 15 we can use TSP lower bounds designed for general metric spaces to answer Question 2.

▶ **Corollary 4.** *Let* $\mathcal{T}$ *be a collection of obstacles in* $\mathbb{R}^3$ *with property* $\mathfrak{P}$ *of total complexity* $m = \text{poly}(|X|)$*. Then* TSP WITH OBSTACLES *has a PTAS if* $\mathfrak{P}$ = *"convex and pairwise disjoint and fat", and it is APX-hard for all other* $\mathfrak{P}$*.*

The PTAS is based on the fact that for $\mathfrak{P}$ = "convex and pairwise disjoint and fat" the free space outside the obstacles has bounded doubling dimension, see Section 2. We can also efficiently compute approximate shortest paths using Har-Peled's algorithm [21], and running any of the known PTASes for TSP in doubling metrics [6, 8] we get the desired algorithm. The lower bound is based on realizing the metric space from the lower bound of Karpinski, Lampis and Schmied [25] using obstacles. The construction in [25] has polynomial spread, which allows us to apply Theorem 3. See Section 5 for a proof of Corollary 4.

1.2 Related work

Motion planning is a central problem in robotics, where the goal is to compute a collision-free path for a set of moving objects (robots) in a space with obstacles. The complexity of the problem depends significantly on the underlying environment and the degrees of freedom

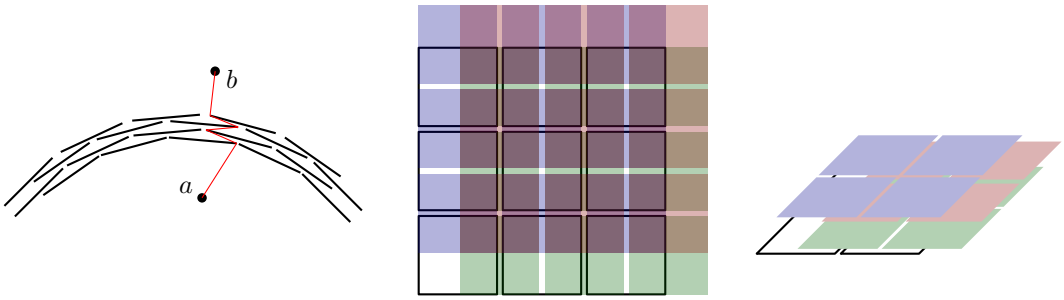


Figure 1 (i) The construction of Dumitrescu and Tóth [15] for a wall in $\mathbb{R}^2$ consisting of convex obstacles. The shortest path from a to b must zigzag between the layers of segments. (ii) The view of a flat wall from $z = \infty$, with 4 shifts (iii) The flat wall construction viewed from a generic point. The planes containing the four types of squares have slightly differing z-coordinates, and they are sandwiched between the planes $z = 0$ and $z = 0.05$.

of the robots. The pioneering works of Schwartz and Sharir [36, 37], and later Canny [10], introduced the first general techniques for solving this problem in the plane. These algorithms have a polynomial dependency on n, the combinatorial complexity of the obstacles, but an exponential dependency on k, the degrees of freedom of the robots. For a detailed discussion of these foundational works, refer to [19, 20]. To highlight the problem's inherent difficulty, note that even in the restricted case of rectangular robots navigating a rectangular environment, the problem is PSPACE-hard [24]. Recently, Agarwal [3] *et al.* presented an FPTAS with a running time of $O(n^2\varepsilon^{-O(1)} \log n)$ for the case of two square robots moving in a polygonal environment. This is the first known PTAS for two robots in such settings.

Focusing now on geodesic distance, many well-studied problems in the Euclidean setting have been explored within polygonal domains under this metric. Notable examples include geodesic convex hulls of points in polygonal domains [7, 39], the 1- and 2-center problems [32, 34], and geodesic spanners [1].

Finally, Abam and Seraji [2] constructed an $8\sqrt{3}$-spanner of size $O(n \log^3 n)$ amid axis-parallel boxes in $\mathbb{R}^3$, which is the only known geodesic spanner result in 3-dimensional space with obstacles.

1.3 Overview of the proof of Theorem 3

Our approach to proving Theorem 3 is inspired by the work of Dumitrescu and Tóth [15], who studied a similar problem in the planar setting. They denote by $\varrho(P)$ the ratio between the geodesic and the Euclidean diameter of a polygon P, and show that over all convex polygons with h convex holes, the supremum of $\varrho(P)$ is between $\Omega(h^{1/3})$ and $O(h^{1/2})$. Crucially, their construction includes building a "wall" of polynomially many short segments that lengthen the geodesic diameter by a large factor, see Figure 1(i) for an illustration.

Our strategy to embed a metric space is as follows. We will define a custom closed surface $\mathcal{S}$ embedded in $\mathbb{R}^3$, and map each point $x_i \in X$ to some point $f(x_i)$ inside. We then place a large collection of obstacles near the surface $\mathcal{S}$ (within distance $1/2$) that ensures that any geodesic that would connect the inside and outside of $\mathcal{S}$ must be very long, longer than the maximum distance in (X, dist_X). This means that the minimum distance between a point $f(x_i)$ and $f(x_j)$ must be realized by a geodesic that stays within $\mathcal{S}$. By setting up $\mathcal{S}$ correctly, we are able to give an upper and lower bound on the length of any geodesic connecting $f(x_i)$ and $f(x_j)$ so that we can ensure $\mathrm{dist}_X(x_i, x_j) \leqslant \mathrm{dist}_{\mathcal{T}}(f(x_i), f(x_j)) \leqslant (1 + \varepsilon) \cdot \mathrm{dist}_X(x_i, x_j)$.

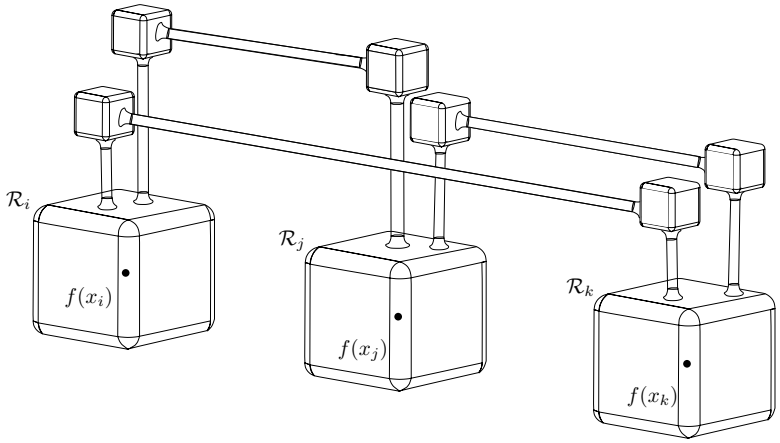


Figure 2 The surface $\mathcal{S}$ for a 3-point metric space. The shortest geodesic from $f(x_i)$ to $f(x_j)$ is forced to go through some tubes, consisting of two vertical and a horizontal cylinder. The length of such a path can be adjusted by changing the length of the vertical cylinders.

Construction of the surface $\mathcal{S}$. Before discussing our obstacle placement, we first describe the construction of $\mathcal{S}$ and the embedding f. (Note that the main text will define the surface $\mathcal{S}$ last.) First, without loss of generality we assume that the minimum distance in (X, dist_X) is at least some large polynomial in $n := |X|$, as we can use a scaling on the final construction to get the desired distance. The function f simply assigns the points x_i to equally spaced points along the x-axis. The surface $\mathcal{S}$ will contain a cube $\mathcal{R}_i$ centered at each point $f(x_i)$.

For each pair i, j we realize the distance of $\mathrm{dist}_X(x_i, x_j)$ via a sequence of three tubes (cylinders), where the first and third cylinder are vertical (parallel to the z-axis) and the second is parallel to the x-axis, see Figure 2. The tubes connect a hole in the top of the cube $\mathcal{R}_i$ to a hole on the cube $\mathcal{R}_j$, and in order to realize the turns, we use two small cubes with holes that connect the first and second as well as the second and third tube in the sequence. Notice that the the length of the second horizontal tube is determined by $f(x_i)$ and $f(x_j)$, thus in order to set the length of the cube sequence, we can vary the length of the two vertical tubes. We place these tubes at different depths (y-coordinates) so that they remain disjoint from each other. One can see that this leads to a good approximate realization, however, this surface needs to be smoother (differentiable) in order to ease the placement of obstacles. Thus $\mathcal{S}$ is defined using differentiable surface patches: we need a rounded cube and a differentiable joint that allows the connection of a flat cube face with a circular cylinder boundary.

A flat wall example. Let us consider an example of a simple placement of convex objects that increases the geodesic distance between the half-space $z < 0$ and the half-space $z > 1$ to significantly more than 1. We call this construction a *flat wall*. In this construction we use squares rather than triangles. Let S_1 be the set of axis-parallel squares of side length 0.99 centered at the points $(a, b, 0.01)$ for $a, b \in \mathbb{Z}$ in the $z = 0.01$ plane. Repeat the same construction 3 more times at heights $z = 0.02, 0.03, 0.04$ by translating S_1 with the vectors $(1/2, 0, 0.01), (1/2, 1/2, 0.02), (0, 1/2, 0.03)$, to get the sets S_2, S_3, and S_4, respectively. See Figure 1(ii) and (iii). Consider a geodesic connecting the planes $z = 0$ and $z = 0.05$: its intersection point p with $z = 0$ will be within ℓ_∞-distance 0.25 to the center of some square in one of the four shifts, and thus it will need to have length at least 0.24 to reach the boundary of this square before it can get to $z = 0.05$. Using further copies of these squares shifted to

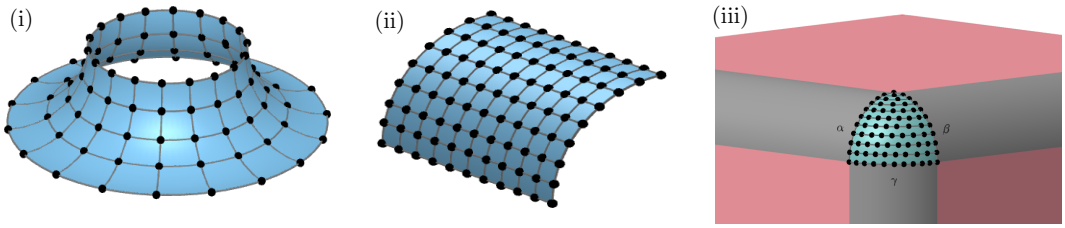


Figure 3 (i) A joint patch with its corresponding net. (ii) A quarter-cylinder patch with its corresponding net. (iii) A spherical triangle patch with its corresponding net.

different height ranges can increase the lower bound of 0.24 as necessary. By making more layers that are packed more densely we can achieve any separation between the halfspaces $z < 0$ and $z > 1$.

While the above construction is quite intuitive, we need a more careful approach to make a similar separation near an embedded closed surface. One can see that the flat construction cannot be adapted to a surface with sharp edges, so we will need to use a smoother closed surface. Even then, we will not be able to place layers of convex objects arbitrarily close together, as convex 2-dimensional shapes will not stay inside a curved surface. The challenge is to keep our objects in different layers disjoint, while creating a strong separation.

Curved wall construction via placing triangle layers. The surface $\mathcal{S}$ that we define has the crucial property that it has offset surfaces (often called parallel surfaces) that are well-defined and differentiable for any distance offset $\delta \in [-1/2, 1/2]$ from $\mathcal{S}$. The offset surface at distance δ is denoted by $\mathcal{S}(\delta)$. Since $\mathcal{S}(\delta)$ is a closed surface, we are able to talk about the inside and outside of $\mathcal{S}(\delta)$, and we can also define the region between two offsets $\mathcal{S}(\delta)$ and $S(\delta')$ when $\delta < \delta'$. In our construction, the offset δ will have a role similar to the z-coordinate in the flat wall construction above: we will place triangles tangent to $S(\delta)$ at some given collection of points. Using a large collection of different offsets, we can ensure the required separation between the inside of $\mathcal{S}(-1/2)$ and the outside of $\mathcal{S}(1/2)$.

We need a handful of geometric properties from $\mathcal{S}(\delta)$ to ensure that we can place our obstacles near it. The most important property is to limit their curvature[1] in some sense. We require that at any point p on $\mathcal{S}(\delta)$ there exist two balls of radius at least $1/2$ whose unique intersection point with $\mathcal{S}(\delta)$ and with each other is p. This helps to ensure that a small 2-dimensional convex shape of diameter $\mu \ll 1$ tangent to $\mathcal{S}(\delta)$ at p stays within distance $O(\mu^2)$ of $\mathcal{S}(\delta)$. In particular, a small triangle will stay between the surfaces $\mathcal{S}(\delta - \mu^2)$ and $\mathcal{S}(\delta + \mu^2)$, and thus triangles of the same size tangent to $\mathcal{S}(\delta - 2\mu^2)$ and $\mathcal{S}(\delta + 2\mu^2)$ will remain disjoint from this triangle.

Finally, we must place the triangles in neighboring layers with "shifts" to ensure that a constant number of consecutive triangular layers of side-length μ triangles creates a separation of at least $\Omega(\mu)$. Here translation is not an option, so we define an (a, b)-*net*, a weaker version of an ε-net (in the metric space sense). For two points $u, v \in \mathbb{R}^3$ we will denote by uv the

[1] While our results could be presented with a more general differential geometric toolbox, the need to construct these (a, b)-nets prompted us to use specific surface patches that are easy to handle algorithmically. One could adapt our techniques to more general smooth embedded surfaces. As a trade-off our surfaces are not smooth (not even twice differentiable) at the shared boundary of our surface patches. We have also used the Euclidean distance function rather than measuring distances within these surfaces to make our construction more elementary and explicit.

line segment between u and v, and by $|uv|$ the length of uv. An (a, b)-*net* of a surface $\mathcal{S}$ embedded in $\mathbb{R}^3$ is a point set $N \subset \mathcal{S}$ where the pairwise distance of points in N is at least a, and for any point $q \in \mathcal{S}$ there is a $p \in N$ such that $|pq| \leqslant b$.

We can show that the offsets of the net points provide a weaker but still useful net in the offset surfaces: a "shared" $(\Omega(\zeta), O(\zeta))$-net can be constructed for the surfaces $\mathcal{S}(\delta)$. Here, "shared" means that we first find a $(\zeta, O(\zeta))$-net on $\mathcal{S}$; sliding every net point along its surface normal gives, on each surface $\mathcal{S}(\delta)$, an $(\Omega(\zeta), O(\zeta))$-net. These net points are then partitioned into constantly many classes, where the distances within points in each class C are large enough so that equilateral triangles of side length $\mu = \Theta(\zeta)$ tangent to $\mathcal{S}(\delta)$ at each point of C will remain disjoint; i.e., these triangles would correspond to a single square set S_1 from our flat wall example. Using all of these partition classes in different offsets $\mathcal{S}(\delta + i\zeta)$ ensures a separation of $\Omega(\zeta)$ for geodesics passing between these layers, as seen with the four shifts of squares in our flat wall example. In our construction we must set ζ small enough to be able to accommodate several iterations of the above construction between $\mathcal{S}(-1/2)$ and $\mathcal{S}(1/2)$, but large enough to get the desired length lower bound for any geodesic going from the inside of $\mathcal{S}(-1/2)$ to the outside of $\mathcal{S}(1/2)$. This concludes the generic construction of the curved wall, as well as the overview of the ideas behind Theorem 3.

2 Doubling dimension among convex fat disjoint obstacles

We denote by $B(p, r)$ the Euclidean ball of radius r centered at $p \in \mathbb{R}^d$, and by $B_{\mathcal{T}}(p, r)$ the geodesic ball of radius r centered at $p \in \mathcal{F}_{\mathcal{T}}$.

We recall the following key result by Chew et al. [11] on convex fat objects. Based on this, we will subsequently show that the free space induced by a set of convex,fat and disjoint obstacles in $\mathbb{R}^d$ has bounded doubling dimension, under the shortest-path metric.

▶ **Lemma 5.** *Let $\mathcal{T}$ be a set of convex, α-fat and disjoint obstacles in $\mathbb{R}^d$. Then, for any $u, v \in \mathcal{F}_{\mathcal{T}}$, we have that $\mathrm{dist}_{\mathcal{T}}(u, v) \leqslant \beta \cdot |uv|$, where β is a constant depending on d and α defined as follows:*

$$\beta = \begin{cases} 1 + \frac{4}{\pi\alpha}, & \text{for } d = 2, \\ 1 + \frac{8d^d}{\alpha}, & \text{for } d \geqslant 3. \end{cases}$$

Since a set $\mathcal{T}$ of convex, fat and disjoint obstacles distorts distances by a constant factor, it follows that the doubling dimension of $(\mathcal{F}_{\mathcal{T}}, \mathrm{dist}_{\mathcal{T}})$ will be bounded; we provide a proof for completeness.

▶ **Lemma 6.** *Let $\mathcal{T}$ be a set of convex, α-fat and disjoint obstacles in $\mathbb{R}^d$. Then the metric space $(\mathcal{F}_{\mathcal{T}}, \mathrm{dist}_{\mathcal{T}})$ has doubling dimension $O(d(d \log d + \log(1/\alpha)))$.*

Proof. Let $B_{\mathcal{T}}(p, r)$ be a geodesic ball centered at $p \in \mathcal{F}_{\mathcal{T}}$. Note that $B_{\mathcal{T}}(p, r) \subseteq B(p, r)$. We will construct a covering of $B_{\mathcal{T}}(p, r)$ by geodesic balls of radius $r/2$. Towards that, we cover $B(p, r)$ by c_d Euclidean balls of radius $r/2$, where $\log(c_d) = \Theta(d)$ is the doubling dimension of $\mathbb{R}^d$ [22, Ch. 10]. Repeating this process recursively $\log(2\beta) + 1$ times, the ball $B(p, r)$ is covered by $c_d^{\log(2\beta)+1}$ Euclidean balls of radius $r/(2^{\log(2\beta)+1}) = r/(4\beta)$. Let $\mathcal{B}$ denote the resulting collection of Euclidean balls. Since $B_{\mathcal{T}}(p, r) \subseteq B(p, r)$, this collection also covers $B_{\mathcal{T}}(p, r)$.

We transform $\mathcal{B}$ into a collection $\mathcal{B}_{\mathcal{T}}$ of geodesic balls in $(\mathcal{F}_{\mathcal{T}}, \mathrm{dist}_{\mathcal{T}})$ of radius $r/2$, such that the union of these geodesic balls covers $B_{\mathcal{T}}(p, r)$. For each Euclidean ball $B(q, r/(4\beta)) \in \mathcal{B}$, we handle the following cases:

1. If $B(q, r/(4\beta)) \cap \mathcal{F}_{\mathcal{T}} = \varnothing$, then we ignore $B(q, r/(4\beta))$ in the cover.
2. If $B(q, r/(4\beta)) \cap \mathcal{F}_{\mathcal{T}} \neq \varnothing$, let $o \in B(q, r/(4\beta)) \cap \mathcal{F}_{\mathcal{T}}$. Include the geodesic ball $B_{\mathcal{T}}(o, r/2)$ in $\mathcal{B}_{\mathcal{T}}$. To verify coverage, note that if $s \in B(q, r/(4\beta)) \cap \mathcal{F}_{\mathcal{T}}$, then:

$$|os| \leqslant \frac{r}{2\beta},$$

By Lemma 5, the geodesic distance between o and s satisfies:

$$\mathrm{dist}_{\mathcal{T}}(o, s) \leqslant \beta \cdot |os| \leqslant \beta \cdot \frac{r}{2\beta} = \frac{r}{2}.$$

Hence, $s \in B_{\mathcal{T}}(o, r/2)$, and $B(q, r/(4\beta)) \cap \mathcal{F}_{\mathcal{T}} \subseteq B_{\mathcal{T}}(o, r/2)$.

Each ball in $\mathcal{B}_{\mathcal{T}}$ has radius $r/2$, and the union of these geodesic balls covers $B_{\mathcal{T}}(p, r)$. The total number of geodesic balls in $\mathcal{B}_{\mathcal{T}}$ is at most $c_d^{\log(2\beta)+1}$. Therefore, the doubling dimension of $(\mathcal{F}_{\mathcal{T}}, \mathrm{dist}_{\mathcal{T}})$ is given by

$$\log(c_d^{\log(2\beta)+1}) = (\log(2\beta) + 1) \cdot \log(c_d) = O(d(d \log d + \log(1/\alpha))).$$ ◀

3 Realizing metric spaces with convex disjoint obstacles

This section contains the main result of our paper. We show that by using only convex and disjoint obstacles in $\mathbb{R}^3$, it is possible to realize any metric space approximately.

3.1 Surface patches

We will consider surfaces embedded in $\mathbb{R}^3$ that are differentiable (i.e., have well-defined normals at each point) described by differentiable surface patches. More precisely, our surface $\mathcal{S}$ will consist of the following types of surface patches:

- Axis-parallel squares of side length at least 1 with 0 or more disjoint circular holes of radius 2, such that each hole is at distance at least 2 from the square, and the hole centers have pairwise distance at least 6. We will refer to such a patch as a *square patch*.
- Sections of a sphere of radius 1 cut out by axis-parallel planes through the center of the sphere. These sections correspond to octants of spheres and will be referred to as *spherical triangle patches*.
- Cylinders of axis length at least 1 and radius 1, or their sections of angle $\pi/2$ around their axes, where the cylinder axis is parallel to some coordinate axis. We will refer to such a patch as a *cylinder patch* and a *quarter-cylinder patch*, respectively.
- *Joints*, which are defined as a surface of revolution obtained by rotating the quarter circular arc $\{(x, y) \mid x^2 + y^2 = 1, x \geqslant 0, y \geqslant 0\}$ around the axis $x = 2$. We only allow rotations where the axis of revolution is parallel to some coordinate axis. We will refer to such a patch as a *joint patch*.

Notice in particular that a joint patch can be used to connect a circular hole of a square patch to a cylinder patch, and the quarter cylinder and spherical triangle patches can be used to "round" an axis parallel cube, i.e., to get the boundary of $C \oplus B_1$, where C is an axis-parallel cube and B_1 is a ball of radius 1, and $\oplus$ denotes the Minkowski sum.

We say that a surface $\mathcal{S}$ is a *patchwork* if it is a closed differentiable surface that is the union of some patches of the above types.

Let $\mathcal{S}$ be a patchwork, and let $\mathbf{n} : \mathcal{S} \to \mathbb{S}^2$ denote its normal bundle. For a fixed $\delta \in (-1/2, 1/2)$ we define the *offset surface of* $\mathcal{S}$ *at distance* δ, denoted by $\mathcal{S}(\delta)$, as $\mathcal{S}(\delta) := \{p + \delta\mathbf{n}(p) \mid p \in \mathcal{S}\}$. Because of the restriction of $|\delta| < 1/2$ we have that $\mathcal{S}(\delta)$ is also a differentiable surface, and in fact it consists of patches of the following types, which we will call δ-patches:

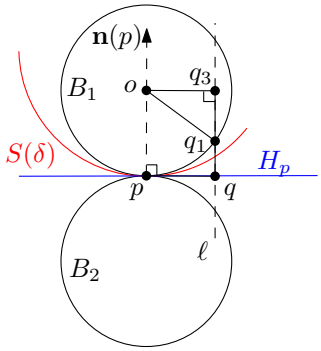


Figure 4 The plane containing a touching segment pq and the normal of $\mathcal{S}(\delta)$ at p. The segment pq will remain very close to $S(\delta)$.

- Square patches (with or without holes of radius 2).
- Sections of a sphere of radius $1+\delta$ cut out by axis-parallel planes through the center of the sphere.
- Cylinders of radius $1+\delta$ or their sections of angle $\pi/2$ around their axes, where the axis is parallel to some coordinate plane.
- Joints, which are obtained by rotating the quarter circular arc $\{(x,y) \mid x^2+y^2 = (1-\delta)^2, x \geqslant 0, y \geqslant 0\}$ around the axis $x=2$; again we only allow rotations where the axis of revolution is parallel to some coordinate axis.

We note that when $\mathcal{S}(\delta)$ is a bounded closed surface of the above type (i.e., it is finite and has no boundary), $\mathbb{R}^3 \backslash \mathcal{S}(\delta)$ consists of two disjoint connected components, one of which is bounded, called *the inside of* $\mathcal{S}(\delta)$, and the other is unbounded, called *the outside of* $\mathcal{S}(\delta)$. Moreover, when $\delta < \delta'$, then $\mathcal{S}(\delta)$ is contained in the inside of $\mathcal{S}(\delta')$, and the set of points *between* $\mathcal{S}(\delta)$ and $\mathcal{S}(\delta')$ are those that are outside $\mathcal{S}(\delta)$ and inside $\mathcal{S}(\delta')$.

We call a differentiable surface $\mathcal{S}$ *r-touchable*[2] if for any $p \in \mathcal{S}$ and any ball B of radius r tangent to $\mathcal{S}$ at p we have that B intersects $\mathcal{S}$ only at p. We say that a patchwork is a *1-patchwork* if it is 1-touchable. Note that if $\mathcal{S}$ is a 1-patchwork then the offsets $\mathcal{S}(\delta)$ of $\mathcal{S}$ are r-touchable for any $r \leqslant 1-|\delta|$. In particular, since we will always have $|\delta| \leqslant 1/2$, all offsets $S(\delta)$ will be 1/2-touchable.

▶ **Lemma 7.** *Let $\mathcal{S}(\delta)$ be an offset of a 1-patchwork surface $\mathcal{S}$. Let H_p be the plane tangent to $\mathcal{S}(\delta)$ at p and let ε be such that $|\delta| < \frac{1}{2} - 2\varepsilon^2$. Then, for any $q \in H_p$ where $|pq| \leqslant \varepsilon$ we have that q lies between $\mathcal{S}(\delta - 2\varepsilon^2)$ and $\mathcal{S}(\delta + 2\varepsilon^2)$.*

Proof. Note that due to the condition $|\delta| < \frac{1}{2} - 2\varepsilon^2$, the offset surfaces $\mathcal{S}(\delta - 2\varepsilon^2)$ and $\mathcal{S}(\delta + 2\varepsilon^2)$ are well-defined. Moreover, we have that $\frac{1}{2} - 2\varepsilon^2 > 0$, or equivalently that $\varepsilon < \frac{1}{2}$.

For the following refer to Figure 4. Consider the tangent balls B_1, B_2 of radius $\frac{1}{2}$ that touch $\mathcal{S}(\delta)$ at p and consider the line ℓ through q parallel to $\mathbf{n}(p)$. Since $|pq| = \varepsilon < \frac{1}{2}$, the line ℓ intersects B_1. Let q_1 denote the point closest to q in $B_1 \cap \ell$. Similarly, ℓ will also intersect B_2 and let q_2 denote the point closest to q in $B_2 \cap \ell$. Because $\mathcal{S}(\delta)$ is 1/2-touchable, we have that the interiors of B_1 and B_2 cannot both be inside or both be outside $\mathcal{S}(\delta)$, thus q_1q_2 intersects $\mathcal{S}(\delta)$. In particular, q_1q or q_2q has to intersect $\mathcal{S}(\delta)$; we assume without loss of generality that q_1q intersects $\mathcal{S}(\delta)$ at a point q^*. Since $|qq^*| < |q_1q|$, it suffices to show that $|q_1q| \leqslant 2\varepsilon^2$.

[2] On a smooth surface this would correspond to bounding the principal curvature in the interval $[-1/r, 1/r]$.

Let o denote the center of B_1 and choose q_3 in the extension of q_1q towards q_1, such that $|q_3q| = \frac{1}{2}$. Then in the right triangle $\triangle oq_1q_3$ we have: $|pq|^2 + \left(\frac{1}{2} - |q_1q|\right)^2 = \left(\frac{1}{2}\right)^2$, which gives: $|q_1q| = \frac{1}{2} - \sqrt{\left(\frac{1}{2}\right)^2 - |pq|^2}$. (The other root of the equation would lead to $|q_1q| > 1/2$, but this is not possible as q_1q is shorter than the radius of B_1.) Thus:

$$|q_1q| = \frac{1}{2} \cdot \left(1 - \sqrt{1 - (2|pq|)^2}\right) \leqslant \frac{1}{2} \cdot (1 - (1 - (2|pq|)^2)) = 2|pq|^2 \leqslant 2\varepsilon^2$$

where in the second inequality above, we have used that $\sqrt{1-x^2} \geqslant 1 - x^2$, for $x \in [0, 1]$. ◀

3.2 Nets, offsets, triangle layers

For the remainder of this section let $\mathcal{S}$ be a 1-patchwork surface, and let $|\delta| < 1/2$. Due to space limit, we provide proofs for the more interesting claims. For the rest of the proofs, refer to the full version of our paper [26]. We start with two technical lemmas that construct nets.

▶ **Lemma 8.** *For any $\zeta \leqslant 1/8$, we can compute a $(\zeta, 8\zeta)$-net of size $O(\text{area}(\mathcal{S})/\zeta^2)$ on $\mathcal{S}$ in $O(\text{area}(\mathcal{S})/\zeta^2)$ time.*

▶ **Lemma 9.** *Let $\zeta \leqslant 1/8$ and let $N \subset \mathcal{S}$ be the $(\zeta, 8\zeta)$-net of $\mathcal{S}$ constructed above. Then, for any $\delta \in (-1/2, 1/2)$, we have that $N_\delta := \{p + \delta\mathbf{n}(p) | p \in N\}$ is a $(\zeta/2, 12\zeta)$-net of $\mathcal{S}(\delta)$.*

▶ **Lemma 10.** *Let N_δ be a $(\zeta/2, 12\zeta)$-net of $\mathcal{S}(\delta)$ for each $|\delta| < 1/2$ constructed above. Then for any fixed δ, any $1 \leqslant t < 1/(4\zeta)$ and any $p \in N_\delta$ we have that the ball $B(p, t\zeta)$ contains at most $O(t^2)$ points of N_δ. Moreover, for any t we can partition N_δ into $O(t^2)$ point sets $N_{\delta,i}$ such that the pairwise distance of points within $N_{\delta,i}$ is at least $t\zeta$.*

Proof. First note that $t \cdot \zeta < 1/4$. Moreover, any cylinder patch of $\mathcal{S}(\delta)$, has distance at least $1/2$ from any square patch of $\mathcal{S}(\delta)$. Therefore, $B(p, t \cdot \zeta)$ cannot intersect both a square and a cylinder patch at the same time. Recall that any two holes on a square patch have pairwise distance at least 6 and are also at distance at least 2 from the boundary of the square patch. Therefore, $B(p, t \cdot \zeta)$ cannot intersect both a quarter-cylinder patch and a joint or a cylinder patch at the same time. The highest number of distinct surface patches $B(p, t \cdot \zeta)$ can intersect is 7, which may happen when it intersects a spherical triangle patch and the three quarter-cylinder patches around it and the three square patches around it. Since it can intersect at most a constant number of distinct patches, it suffices to show that it can contain at most $O(t^2)$ points of N_δ from each different type of surface patch.

To that end, let $\mathcal{P}$ denote a patch which is not a square patch and not a spherical triangle patch. For the remaining types of patches, the constructed net is taken as a point set of the form $I(A, B)$, where elements in A and B are quarter-circular arcs, line segments or circles. In all cases, we have argued already that any two elements in A have distance at least $\zeta/2$ (since now N_δ is a $(\zeta/2, 12\zeta)$-net) and any two elements in B have also distance at least $\zeta/2$. Therefore, from any point $p \in \mathcal{P}$ and within distance $t \cdot \zeta$, we can reach at most $2t$ elements of A and $2t$ elements of B. Therefore we can reach at most $4t^2$ net points of $\mathcal{P}$. On a spherical triangle patch, the arcs of A again have pairwise distance at least $\zeta/2$, so we can reach at most $O(t)$ arcs, and on each arc we can reach at most $O(t)$ points as on each quarter-circular arc the distance between consecutive points is at least $\zeta/2$.

For a square patch, the process of construction was slightly different due to the existence of the holes. Using the same argumentation as above, we can deduce that $B(p, t \cdot \zeta)$ can contain at most $4t^2$ net points from the initially placed grid. Moreover, $B(p, t \cdot \zeta)$ can intersect

at most one hole. Note that a single hole contains $O(1/\zeta)$ net points and since $t < 1/(2\zeta)$, it contains at most $O(t)$ net points. Therefore, $B(p, t \cdot \zeta)$ can contain at most $5t^2$ net points of a square patch.

Since we argued that $B(p, t \cdot \zeta)$ can intersect at most 7 distinct surface patches, we have that it can contain at most $7 \cdot 5t^2 = 35t^2$ points of N_δ.

For the second part of the lemma create $b = \lfloor 35t^2 + 1 \rfloor$ partition classes $N_{\delta,1}, N_{\delta,2}, ..., N_{\delta,b}$ and greedily place points in these classes: we place each point $p \in N_\delta$ in the available class of smallest index (that is, a class of minimum index not already containing any point q such that $|pq| < t \cdot \zeta$). Note that in this way, there will always be a class which is available, since for any point there are at most $35t^2$ points that cannot be in the same class with it. ◀

▶ **Lemma 11.** *There exists a constant ν such that for any $\zeta < 1/(2\nu)$ and $-1/2 < \delta < 1/2 - \nu\zeta^2$ there is collection $\mathcal{T}$ of $O(\text{area}(\mathcal{S})/\zeta^2)$ congruent equilateral triangles of side length $\Theta(\zeta)$ that satisfy the following properties:*

- **(i)** *each triangle in $\mathcal{T}$ is located between $\mathcal{S}(\delta)$ and $\mathcal{S}(\delta + \nu\zeta^2)$*
- **(ii)** *the triangles in $\mathcal{T}$ are pairwise disjoint, and*
- **(iii)** *the geodesic distance from any point of $\mathcal{S}(\delta)$ to any point of $\mathcal{S}(\delta + \nu\zeta^2)$ is at least 4ζ.*

Proof. By applying Lemma 8 and Lemma 9 for we obtain a $(\zeta/2, 12\zeta)$-net N_δ of $\mathcal{S}(\delta)$ of size $O(\text{area}(\mathcal{S})/\zeta^2)$. We set $t = 48(1 + \sqrt{3})$, and choose ν large enough such that $\zeta < 1/(4t)$. Then, by Lemma 10, we can partition N_δ into $b = \lfloor 35t^2 + 1 \rfloor$ classes $(N_{\delta,i})_{i=1}^b$, where points within each class $N_{\delta,i}$ have pairwise distance at least $t \cdot \zeta = 48(1 + \sqrt{3})z$. We will now place each class $N_{\delta,i}$ on a different offset $\mathcal{S}(\delta_i)$, defined as follows. We start by setting $\nu = 4b + 1$ and define $\delta_i := \delta + 4i \cdot \zeta^2$, for $i = 1, 2, .., b$. From now on, we consider the points of each class $N_{\delta,i}$ on the surface $\mathcal{S}(\delta_i)$ defined as above.

The above definition ensures $\delta_i \leqslant \delta + 4b\zeta^2 < \delta + \nu\zeta^2 < 1/2$. Thus, the offsets $S(\delta_i)$ are well-defined. Furthermore, we also have the following:

$$\text{For } 1 \leqslant i \leqslant b, \text{ we have that } \delta_i - 2\zeta^2 < \delta_i < \delta_i + 2\zeta^2 < \delta_{i+1}. \tag{1}$$

For $1 \leqslant i \leqslant b$ and $p \in N_{\delta,i}$, let T_p denote the equilateral triangle of side length $48\sqrt{3}\zeta$ centered at p, which lies on the tangent plane H_p at p. Let $\mathcal{T}_i = \{T_p : p \in \mathcal{N}_{\delta,i}\}$ and $\mathcal{T} = \bigcup_{i=1}^b \mathcal{T}_i$. We now argue that $\mathcal{T}$ satisfies the three properties of the Lemma:

- **(i)** This holds trivially since $\delta + 4\zeta^2 \leqslant \delta_i \leqslant \delta + 4b\zeta^2 < \delta + \nu\zeta^2$.
- **(ii)** Lemma 7 together with (1), ensure that any two triangles in $\mathcal{T}$ that belong to different offsets are disjoint. Next, we argue that any two triangles $T_p, T_q \in \mathcal{T}_i$ are disjoint. From Lemma 10 we know that $|pq| \geqslant 48(1 + \sqrt{3})\zeta > 96\zeta$. Since each of our triangles has circumradius 48ζ, it follows that T_p, T_q cannot intersect.
- **(iii)** Refer to Section 3.2 for an illustration. Let $p \in \mathcal{S}(\delta)$ and $q \in \mathcal{S}(\delta + \nu\zeta^2)$ and consider a shortest geodesic path $\pi(p, q)$. We want to show that $\text{dist}_{\mathcal{T}}(p, q) \geqslant c \cdot \zeta$ for some constant c. Let u be a point in $\pi(p, q) \cap \mathcal{S}(\delta_1)$ and consider the segment s between the points $u - \mathbf{n}(u)$ and $u + \mathbf{n}(u)$. We consider the following two cases:
  - *Case I: There exists $v \in \pi(p, q)$, such that $\text{dist}_{\mathbb{R}^3}(v, s) \geqslant 4\zeta$.*
    Then π has the desired length.
  - *Case II: $\pi(p, q)$ is contained in the cylinder with axis s and radius 4ζ*
    Let $\mathcal{C}(ab, x)$ denote the cylinder of axis ab and radius x. We will show that there exists $T \in \mathcal{T}$ which cuts the cylinder $\mathcal{C}(s, 4\zeta)$ in such a way that the top and bottom disk of the cylinder are completely contained in different sides of the cut. Since $\pi(p, q)$ connects these disks and it is disjoint from all triangles $T \in \mathcal{T}$, this will give a contradiction.

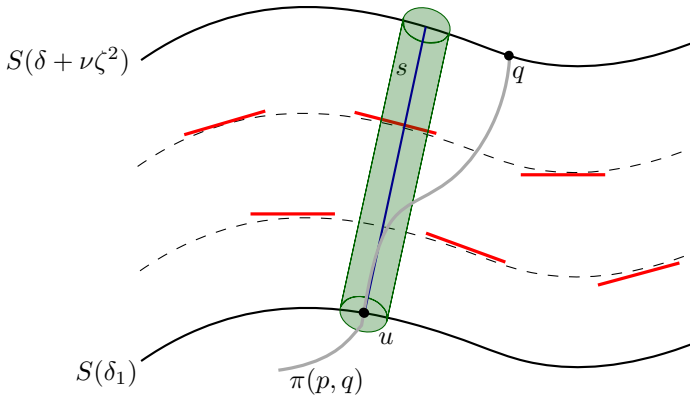


Figure 5 Illustration for the proof of Lemma 11 (side view). The geodesic $\pi(p, q)$ is shown in grey. Note that the cylinder around s is cut by one of the red triangles, therefore $\pi(p, q)$ needs to exit the cylinder.

By the properties of N_δ, we know that there exists a surface $\mathcal{S}(\delta_i)$, such that $s \cap \mathcal{S}(\delta_i)$ is within distance 12ζ from a point $w \in N_{\delta,i}$. We will argue that T_w is the required triangle. Towards that, let $s_i = s \cap \mathcal{S}(\delta_i)$ and recall that T_w has inradius 24ζ and denote by D the incircle of T_w. Observe that $\mathcal{C}(s_{i-1}s_{i+1}, 4\zeta) \subset B(w, 24\zeta)$. By Lemma 7, we know that T_w is between $\mathcal{S}(\delta_{i-1})$ and $\mathcal{S}(\delta_{i+1})$. Since D splits $B(w, 24\zeta)$ in two pieces where s_{i-1} and s_{i+1} are in different parts, we get that D has to cut $\mathcal{C}(s_{i-1}s_{i+1}, 4\zeta)$. We conclude that indeed T_w cuts $\mathcal{C}(s, 4\zeta)$ in the described manner, which is a contradiction. ◀

By iterating the construction of Lemma 11 we get our curved wall separation theorem.

▶ **Theorem 12** (Polynomial separation). *Let $\mathcal{S}$ be a 1-patchwork surface. Then for any $\sigma > 1$ there is a collection of $O(\sigma^4 \text{area}(\mathcal{S}))$ pairwise disjoint congruent equilateral triangle obstacles between $\mathcal{S}(-1/2)$ and $\mathcal{S}(1/2)$, such that the geodesic distance from any point of $\mathcal{S}(-1/2)$ to any point of $\mathcal{S}(1/2)$ is at least σ.*

3.3 The construction and the proof of Theorem 3

Recall that (X, dist_X) is the metric space whose distances we want to realize, with $|X| = n$ and spread $\Phi = \max_{a,b,c,d \in X}(\frac{\text{dist}_X(a,b)}{\text{dist}_X(c,d)})$, and $\varepsilon > 0$ denotes an accuracy parameter. We can assume that the minimum distance of any two points in X is $41n^3/\varepsilon$, as in general we can scale the whole construction so that the minimum geodesic distance exactly matches the minimum distance in X. We construct a 1-patchwork surface $\mathcal{S}$ together with an injection $f : X \to \mathbb{R}^3$ as follows.

For each point $x_i \in X$, $1 \leqslant i \leqslant n$, let $f(x_i) = (40n^2 \cdot i, 0, -10n^2)$. Let $\mathcal{C}_i$ denote the cube centered at $f(x_i)$ of side-length $20n^2 - 2$. We define the rounded cube $\mathcal{R}_i = \mathcal{C}_i \bigoplus B_1$, where B_1 denotes the ball of radius 1. Note that each rounded cube $\mathcal{R}_i$ has *size* $20n^2$, where we define the size of a rounded cube to be the side-length of its bounding box. Then the top square patch of each $\mathcal{R}_i$ lies on the plane $z = 0$, has side length $20n^2 - 2$ and is centered at the point $(40n^2 \cdot i, 0, 0)$. For each pair (i, j) where $i < j$, we model the distance between x_i and x_j as follows. We start by placing two holes $H_{i,j}, H_{j,i}$ on the top square patch of $\mathcal{R}_i$ and $\mathcal{R}_j$ respectively, such that:

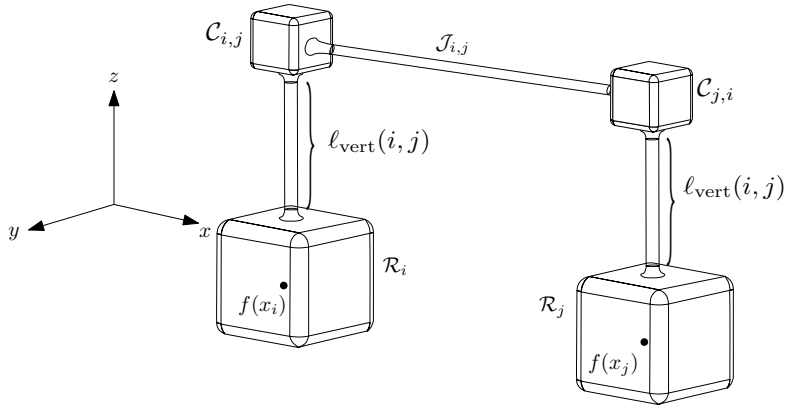


Figure 6 A tube consisting of three cylinders and a turn that connects the rounded cubes $\mathcal{R}_i$ and $\mathcal{R}_j$. We will set $\ell_{\text{vert}}(i,j)$ to ensure that the length of the tube corresponds to the distance $\text{dist}_X(x_i, x_j)$.

- $H_{i,j}$ has radius 2 and center $(40n^2 \cdot i, -10n^2 + 10(n-1) \cdot i + 10j - 10, 0)$, and
- $H_{j,i}$ has radius 2 and center $(40n^2 \cdot j, -10n^2 + 10(n-1) \cdot i + 10j - 10, 0)$.

Observe that the centers of any two holes $H_{i,j}, H_{j,i}$ defined this way have the same y-coordinates. Moreover, the y-coordinates of the centers of all holes are in the range $[-10n^2 + 10n, -10n]$. Since the y-coordinates of the top square patches are in the range $[-10n^2 + 1, 10n^2 - 1]$, this ensures that:

1. holes are well-defined,
2. each hole has distance at least 2 from the boundary of its square patch, and
3. for each $1 \leqslant i, j, k \leqslant n$, where i, j, k are pairwise different, the centers of $H_{i,j}$ and $H_{i,k}$ have distance at least 10. As a result, $H_{i,j}$ and $H_{i,k}$ have distance at least 6, as required in the definition of a square patch.

It will be useful for the rest of the description, to define a *cylinder-joint patch of length* ℓ as a cylinder patch of length ℓ, with two joint patches attached to its bases. Note that a cylinder-joint patch can be used to connect two holes that are opposite to each other.

Next, for each $1 \leqslant i < j \leqslant n$, we connect $H_{i,j}$ and $H_{j,i}$ as follows. We attach to both $H_{i,j}$ and $H_{j,i}$ a cylinder-joint patch of length $\ell_{\text{vert}}(i,j)$, which we denote by $\mathcal{K}_{i,j}$ and $\mathcal{K}_{j,i}$ respectively. The length $\ell_{\text{vert}}(i,j)$ will depend on ε and its exact value will be set in the proof of Theorem 3. We then attach $\mathcal{K}_{i,j}$ to a rounded cube $\mathcal{C}_{i,j}$ of size 8, through a circular hole of radius 2 centered at the center of the bottom square patch of $\mathcal{C}_{i,j}$. Note that this hole has distance at least 2 from the boundary of the square patch. Similarly, we attach $\mathcal{K}_{j,i}$ to a rounded cube $\mathcal{C}_{j,i}$ of size 8, through a circular hole of radius 2 centered at the center of the bottom square patch of the cube. We finally connect $\mathcal{C}_{i,j}$ and $\mathcal{C}_{j,i}$ through a horizontal cylinder-joint patch of length $\ell_{\text{hor}}(i,j) = 40n^2 \cdot (j-i)$, denoted by $\mathcal{J}_{i,j}$. This concludes the construction of $\mathcal{S}$. Refer to Figure 6 for an illustration of the connection between $f(x_i), f(x_j)$. After constructing $\mathcal{S}$ we apply Theorem 12 for $\sigma := \max_{x_i,x_j} \text{dist}_X(x_i, x_j) = O(\Phi \cdot n^3/\varepsilon)$, to achieve a separation of $\max_{x_i,x_j} \text{dist}_X(x_i, x_j)$ between $\mathcal{S}(-1/2)$ and $\mathcal{S}(1/2)$. We let $\mathcal{T}$ denote the resulting set of triangle obstacles.

Next, we sketch the proof of Theorem 3. Refer to the full version [26] for a detailed proof.

Proof sketch. We need to upper and lower bound the length of a geodesic from $f(x_i)$ to $f(x_j)$. The choice of σ ensures that we only need to consider geodesics within $\mathcal{S}(1/2)$. We set $\ell_{\text{vert}}(i,j)$ so that any geodesic that passes through the three-cylinder-tube connecting

$\mathcal{R}_i$ and $\mathcal{R}_j$ has length at least $\mathrm{dist}_X(x_i, x_j)$. Now a geodesic passing through multiple such tubes will have a length that is at least the sum of the corresponding distances, thus by the triangle inequality in X any such path will also have length at least $\mathrm{dist}_X(x_i, x_j)$.

To upper bound the geodesic distance of $f(x_i)$ to $f(x_j)$, we observe that there is a direct path in $\mathcal{S}(-1/2)$ connecting $f(x_i)$ and $f(x_j)$ through the designated three-cylinder tube. This will contain some additional short segments compared to our lower bound, namely, the segments connecting the endpoints of the geodesic ($f(x_i)$ and $f(x_j)$) to the entrance and exit of the tube. The total length of these segments is shown to be less than $\varepsilon \mathrm{dist}_X(x_i, x_j)$. ◀

We note the following observation from within the proof of Theorem 3 for later use.

▶ **Observation 13.** *There is a path connecting $f(x_i)$ and $f(x_j)$ in the inside of $\mathcal{S}(-1/2)$ that has length at most $(1+\varepsilon) \cdot \mathrm{dist}_X(x_i, x_j)$.*

▶ Remark 14 (Exact realizations). It is particularly challenging to realize a metric *exactly* with convex obstacles. While our construction can be modified to ensure the exact length of the shortest path through the corresponding tube has the same exact length as the original distance in (X, dist_X), this is not sufficient.

The problem can be seen by considering the metric induced by a star on 4 vertices. If a, b, c are the leaves and x is the center of the star, then the the shortest $f(a) \to f(b)$ path will be realized by going through the $f(a) \to f(x)$ tube and the $f(x) \to f(b)$ tube, foregoing the small detour to $f(x)$ within its rounded cube. This path will be strictly shorter than $\mathrm{dist}_{\mathcal{T}}(f(a), f(x)) + \mathrm{dist}_{\mathcal{T}}(f(x), f(b))$. While a realization exists for star graph metrics using relatively open triangles, this does not generalize. Can we for example realize (X, dist_X) exactly with disjoint convex obstacles assuming the image of dist_X is $\{1, 2\}$?

4 Realizing metrics with fat obstacles

We tweak our construction to prove the following theorem and answer Question 1 for convex fat non-disjoint obstacles and for fat, disjoint but non-convex obstacles.

▶ **Theorem 15.** *Let $\varepsilon \in (0, 1)$ and let (X, dist_X) be a metric space of size $|X| = n$ and spread $\Phi := \max_{a,b,c,d \in X} \left(\frac{\mathrm{dist}_X(a,b)}{\mathrm{dist}_X(c,d)}\right)$. Then there exists a collection $\mathcal{T}$ of obstacles for each of the following restrictions:*

- **(i)** *$\mathcal{T}$ consists of $O(n^{17}\Phi^5/\varepsilon^5)$ congruent regular simplices (that are not necessarily disjoint).*
- **(ii)** *$\mathcal{T}$ consists of $O(n^5\Phi/\varepsilon)$ disjoint, similarly-sized α-fat objects that are homeomorphic to a ball. The objects have total complexity $O(n^{17}\Phi^5/\varepsilon^5)$, where $\alpha > 0$ is a constant.*

In both cases, there is an injection $f : X \to \mathbb{R}^3$ such that

$$\mathrm{dist}_X(a, b) \leqslant \mathrm{dist}_{\mathcal{T}}(f(a), f(b)) \leqslant (1+\varepsilon) \cdot \mathrm{dist}_X(a, b).$$

for all $a, b \in X$. The set of obstacles can be constructed in $\mathrm{poly}(n, \Phi, 1/\varepsilon)$ time.

Proof. (i) We modify the set of triangular obstacles in our basic construction as follows. Recall from the proof of Lemma 11 that each triangle $T \in \mathcal{T}$ is tangent at the center of T to some $\mathcal{S}(\delta)$ at a point $p \in \mathcal{S}(\delta)$, for some $-1/2 < \delta < 1/2$. We turn T into a regular tetrahedron T' where one face of T' is T, and the vertex of T' outside T is placed along the normal ray from p, that is, the tetrahedron will point to the outside of $\mathcal{S}(\delta)$. The obstacles in the new collection obtained this way, denoted by $\mathcal{T}'$, are convex and 1/3-fat.

We can now prove Theorem 3 for $\mathcal{T}'$ instead of $\mathcal{T}$. For the lower bound, it suffices to observe that $\mathcal{F}_{\mathcal{T}'} \subset \mathcal{F}_{\mathcal{T}}$. This implies that the set of geodesics in $\mathcal{F}_{\mathcal{T}'}$ is a subset of the geodesics in $\mathcal{F}_{\mathcal{T}}$, and thus for any two points $x_i, x_j \in X$ we have $\mathrm{dist}_{\mathcal{T}'}(f(x_i), f(x_j)) \geqslant$

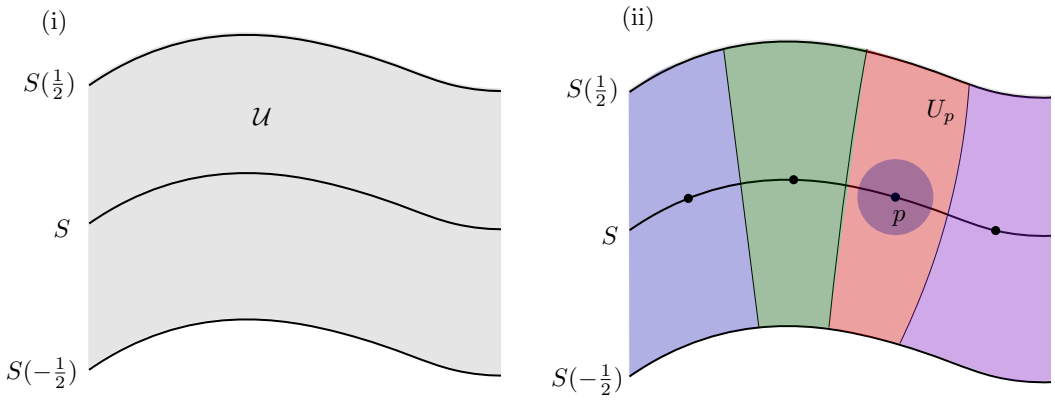


Figure 7 (i) Side view of the region $\mathcal{U}$ (shaded in grey). (ii) The region $U_p = C_p \cap \mathcal{U}$ (in red), together with the ball $B_p \subset U_p$ (in blue), indicating that U_p is fat.

$\text{dist}_{\mathcal{T}}(f(x_i), f(x_j)) \geqslant \text{dist}_X(x_i, x_j)$. On the other hand, the inside of $\mathcal{S}(-1/2)$ is in $\mathcal{F}_{\mathcal{T}'}$, due to the way we constructed the tetrahedra (pointing "out" of the offset surface they are tangent to). Observation 13 now implies that $\text{dist}_{\mathcal{T}'}(f(x_i), f(x_j)) \leqslant (1+\varepsilon)\text{dist}_X(x_i, x_j)$.

(ii) Let $\mathcal{T}$ be the triangle collection constructed by Section 3.3, and let $\mathcal{U}$ be the union of all offsets of $\mathcal{S}$, that is, $\mathcal{U} := \bigcup_{\delta\in[-1/2,1/2]} \mathcal{S}(\delta)$. Refer to Figure 7 for an illustration. Let N be a $(1/8, 1)$-net of $\mathcal{S}$ obtained from Lemma 9 with $\zeta = 1/8$. Consequently, $|N| = \text{area}(\mathcal{S}) = O(n^5\Phi/\varepsilon)$. Compute the *Voronoi diagram* of N in $\mathbb{R}^3$, that is, a partition of $\mathbb{R}^3$ into cells according to the closest point of N. The Voronoi cells can be computed in $\text{poly}(|N|)$ time [18, Ch. 27]. For the cell C_p of $p \in N$ let $U_p := C_p \cap \mathcal{U}$; note that each U_p can be represented explicitly in $\text{poly}(|N|) = \text{poly}(|\mathcal{T}|)$ space. Observe that U_p has diameter at most $\text{diam}(U_p) \leqslant 3/2$: indeed, any point in $\mathcal{U}$ is within distance at most $1/2$ from $\mathcal{S}$ (along the normal direction or its inverse), and any point of $\mathcal{S}$ is within distance at most 1 from some point of N by the net property of N.

We claim that each set U_p contains a ball B_p of radius $1/16$. Clearly the Voronoi cell C_p contains this ball by the net property; it remains to show that B_p is contained in $\mathcal{U}$. Consider the line ℓ normal to $\mathcal{S}$ at p, and let $p(\delta)$ be the point of this line on $\mathcal{S}(\delta)$ where $\delta \in [-1/16, 1/16]$. Notice that a point $q \in B_p$ the nearest point of ℓ is some point $p(\delta)$, and the tangent plane H_δ of $\mathcal{S}(\delta)$ at $p(\delta)$ contains q. Since $|q, p(\delta)| \leqslant 1/16$, we have by Lemma 7 that q is between $\mathcal{S}(\delta - 2/16^2)$ and $\mathcal{S}(\delta + 2/16^2)$. Since $\delta < 1/16$, this implies that $q \in \mathcal{U}$, and concludes the proof that $B_p \subset U_p$.

Consider now the obstacle set $\mathcal{T}$, and notice that its triangles have positive pairwise distance from each other and also from $\mathcal{S}(-1/2)$ and from $\mathcal{S}(1/2)$. Let $\mu > 0$ be some small number such that the Euclidean μ-neighborhoods of the triangles of $\mathcal{T}$ remain pairwise disjoint, and they remain subsets of $\mathcal{U}$. For a triangle $T \in \mathcal{T}$ let $x(T)$ denote the center of T and let T^μ denote the set of points q in $\mathbb{R}^3$ such that $\text{dist}_{\mathbb{R}^3}(q, T) < \mu$ (i.e., T^μ is an open set). Here $\text{dist}_{\mathbb{R}^3}(q, T)$ denotes the minimum Euclidean distance between q and a point of T.

Let $U_p^{-\mu}$ denote the set of points q with $\text{dist}_{\mathbb{R}^3}(q, \mathbb{R}^3 \setminus U_p) \geqslant \mu$, that is, $U_p^{-\mu}$ is a closed subset of U_p where a small neighborhood of the boundary of U_p has been removed. Consequently, the sets $U_p^{-\mu}$ are pairwise disjoint.

We group $\mathcal{T}$ according to the location of their triangle centers. Let $\mathcal{T}_p$ denote the set of triangles $T \in \mathcal{T}$ where $x(T) \in U_p$; if $x(T)$ is on the shared boundary of several sets $(U_p)_{p\in N}$, then we assign T to one of the corresponding classes arbitrarily. As a result $\{\mathcal{T}_p\}_{p\in N}$ is a partition of $\mathcal{T}$. For each point $p \in N$ we define the following obstacle W_p by adding all triangles of $\mathcal{T}_p$ to $U_p^{-\mu}$, but subtracting T^μ for all $T \in \mathcal{T} \setminus \mathcal{T}_p$:

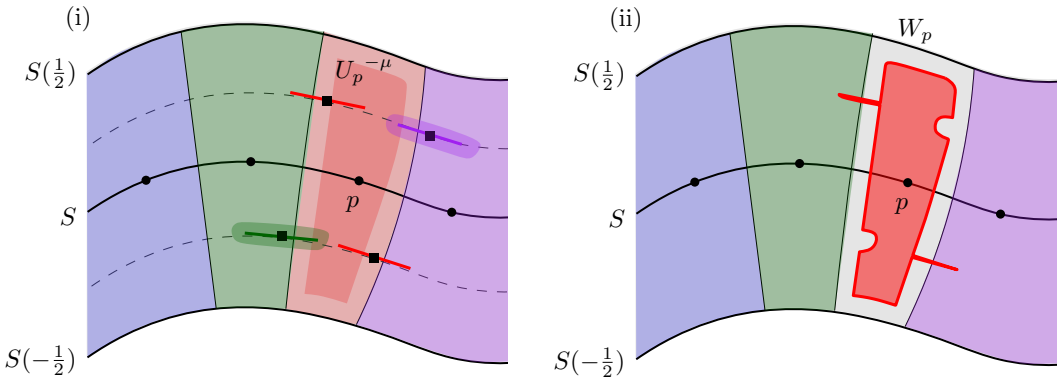


Figure 8 (i) By shrinking U_p, we obtain the region $U_p^{-\mu}$. Square points correspond to centers of triangles in $\mathcal{T}$. Four triangles are drawn, again in side view. For illustration purposes, various objects have not been drawn to scale. (ii) The final obstacle W_p, obtained by attaching to $U_p^{-\mu}$ the two red triangles, and removing a small neighborhood of the other two.

$$W_p := \left(U_p^{-\mu} \cup \left(\bigcup_{T\in\mathcal{T}_p} T\right)\right) \setminus \left(\bigcup_{T\in\mathcal{T}\setminus\mathcal{T}_p} T^{\mu}\right).$$

Refer to Figure 8 for an illustration of an obstacle W_p. The sets W_p are computed in polynomial time, and they are closed sets that are pairwise disjoint. Moreover, each set W_p has diameter at most $\operatorname{diam}(W_p) \leqslant \operatorname{diam}(U_p) + s_T < 2$ where $s_T < 1/100$ is the side length of the triangles in T. Recall that each set U_p contains a ball of radius $1/16$ centered at p. It follows that W_p contains the ball of radius $1/16 - \mu - s_T > 1/32$ centered at p. We conclude that the sets W_p are $1/64$-fat.

The proof that the obstacle set $\{W_p\}_{p\in N}$ approximately realizes $(X, \operatorname{dist}_X)$ can be argued as in part (i). The number of obstacles is $|N| = O(n^5\Phi/\varepsilon)$. ◀

We note that both the size of our obstacle collection and the construction time could be further improved; here, however, we did not attempt to optimize for these aspects.

5 Algorithms and lower bounds for TSP WITH OBSTACLES

This section is devoted to proving Corollary 4. In order to give an algorithm, we will first need to (approximately) compute the distances between our input points. Let $\mathcal{T}$ denote a collection of pairwise-disjoint polyhedral obstacles in $\mathbb{R}^3$ of total complexity m. Har-Peled [21] showed how to construct an ε-approximate shortest path map from any source point $s \in \mathcal{F}_{\mathcal{T}}$, in time roughly $O(m^4/\varepsilon^6)$. From his result we can deduce the following:

▶ **Theorem 16** (Har-Peled [21]). *Let $\mathcal{T}$ be a collection of pairwise-disjoint polyhedral obstacles in $\mathbb{R}^3$ of total complexity m and let $S \subset \mathcal{F}_{\mathcal{T}}$, be a set of $|S| = n$ points. Then, for any $0 < \varepsilon < 1$, we can compute a $(1+\varepsilon)$-approximation for the all-pairs-shortest-paths problem on S in time:*

$$O\left(n\frac{m^4}{\varepsilon^2}\left(\frac{\beta(m)}{\varepsilon^4}\log\frac{m}{\varepsilon} + \log(m\rho)\log(m\log\rho)\right)\log\frac{1}{\varepsilon}\right)$$

where ρ is the ratio of the longest edge in $\mathcal{T}$ with the distance of the closest pair of points in S, and $\beta(m) = \alpha(m)^{O(\alpha(m))^{O(1)}}$, and $\alpha(m)$ is the inverse of the Ackermann function.

Note that Har-Peled's result is based on constructing a graph whose vertices correspond to points and whose edges correspond to pairwise visible point pairs, where edge lengths are high-precision estimates of the distance of the visible point pair. Har-Peled's result computes the true shortest paths in this graph. Consequently, the resulting approximate distance function $\mathrm{dist}_{\mathrm{approx}}$ on S forms a metric space. One can also show using arguments similar to Lemma 6 that for any $\varepsilon \in (0,1)$ the doubling dimension of $(S, \mathrm{dist}_{\mathrm{approx}})$ is at most constant times the doubling dimension of $(S, \mathrm{dist}_{\mathcal{T}})$.

We now mention the result by Banerjee, Bartal, Gottlieb and Hovav, on TSP in doubling spaces [6].

▶ **Theorem 17** (Corollary 35 in [6]). *Let (X, dist_X) be a metric space with $|X| = n$ points and of bounded doubling dimension δ. A $(1+\varepsilon)$-approximation to the optimal tour of X can be computed by a randomized algorithm in $2^{(\delta/\varepsilon)^{O(\delta)}} n + (1/\varepsilon)^{O(\delta)} n \log n$ time.*

By combining Theorem 17 and Theorem 16, we will obtain a PTAS for the case $\mathfrak{P} =$ convex $\wedge$ pairwise disjoint $\wedge$ fat. For the other cases, the APX-hardness is based on the inapproximability result by Karpinski, Lampis and Schmied on Metric TSP [25]. They showed an inapproximability bound of 123/122. By inspection of the construction given in [25], we have the following crucial observation:

▶ **Observation 18.** *It is NP-hard to approximate* Metric TSP *within a ratio of* 123/122*, even when the underlying metric space has polynomial spread.*

Proof. This follows by inspection of the construction by Karpinski, Lampis and Schmied (Section 5.1 in [25]), which is based on a the shortest path metric of a connected edge-weighted graph. The weights of all edges used are small constants. Namely, the smallest edge has weight 0.5, and the longest edge has length 2. Therefore, the graph has diameter at most $2(n-1)$, and the construction has a spread at most $\frac{2(n-1)}{0.5} = 4(n-1) = O(n)$. ◀

Now we have all the necessary elements to prove Corollary 4.

▶ **Corollary 4.** *Let $\mathcal{T}$ be a collection of obstacles in $\mathbb{R}^3$ with property $\mathfrak{P}$ of total complexity $m = \mathrm{poly}(|X|)$. Then* TSP with obstacles *has a PTAS if $\mathfrak{P}$ = "convex and pairwise disjoint and fat", and it is APX-hard for all other $\mathfrak{P}$.*

Proof of Corollary 4. Let $S \subset \mathcal{F}_{\mathcal{T}}$ denote a set of n points, whose optimal tour we want to compute. We have the following cases:

- **The obstacles in $\mathcal{T}$ are convex, α-fat and pairwise disjoint.**
  We show that then TSP with obstacles admits a PTAS. Towards that, let $\varepsilon \in (0,1)$. By Theorem 16, we can approximate $\mathrm{dist}_{\mathcal{T}}$ up to a factor of $1+\varepsilon/3$. Let $\mathrm{dist}_{\mathrm{approx}}$ denote the approximate metric. The doubling dimension of $(S, \mathrm{dist}_{\mathrm{approx}})$ is at most a constant factor larger than that of $(S, \mathrm{dist}_{\mathcal{T}})$. Therefore, by Lemma 6, it is at most $O(1)$. By Theorem 17, we can get a $(1+\varepsilon/3)$-approximation for TSP with obstacles on $(S, \mathrm{dist}_{\mathrm{approx}})$, with running time $2^{1/\varepsilon^{O(1)}} n + 1/\varepsilon^{O(1)} n \log n$. Since $(1+\varepsilon/3)^2 < 1+\varepsilon$, the result is a $(1+\varepsilon)$-approximation for TSP with obstacles.
- **The obstacles in $\mathcal{T}$ are convex and pairwise disjoint but not fat, or fat and convex but non-disjoint, or fat and disjoint but non-convex.**
  We will show that in this case the problem is APX-hard, with an inapproximability bound of $\frac{123}{122} - \varepsilon$, for any $\varepsilon > 0$. For a set $S \subset \mathcal{F}_{\mathcal{T}}$, let $\mathrm{OPT}_{\mathcal{T}}(S)$ denote the length of an optimal tour through S. Assume that we could, in polynomial time, compute a tour of length $T_{\mathrm{approx}}(S)$ such that $T_{\mathrm{approx}}(S) \leqslant \left(\frac{123}{122} - \varepsilon\right) \mathrm{OPT}_{\mathcal{T}}(S)$, for a fixed $\varepsilon > 0$.

Let (X, dist_X) denote an instance of Metric tsp, where $|X| = n$ and X has spread $\mathrm{poly}(n)$. From Observation 18, we have a 123/122 inapproximability bound for computing the optimal tour on X, denoted by $\mathrm{OPT}(X)$. From Theorem 3 or Theorem 15, we compute a set of obstacles $\mathcal{T}$ in $\mathbb{R}^3$ and an injection $f : X \to \mathbb{R}^3$, such that:

$$\mathrm{OPT}_{\mathcal{T}}(f(X)) \leqslant (1 + \varepsilon')\mathrm{OPT}(X),$$

where we set ε' to satisfy $\frac{123}{122(1+\varepsilon')} = \frac{123}{122} - \varepsilon$.
By our assumption, we can compute a value $T_{\mathrm{approx}}(f(X))$ such that:

$$T_{\mathrm{approx}}(f(X)) \leqslant \left(\frac{123}{122} - \varepsilon\right) \mathrm{OPT}_{\mathcal{T}}(f(X))$$

Therefore, $T_{\mathrm{approx}}(f(X)) \leqslant \frac{123}{122}\mathrm{OPT}(X)$, which contradicts the APX-hardness of Metric tsp. ◀

6 Conclusion

In this paper we have shown that all metric spaces are approximately realizable in $\mathbb{R}^3$ using convex disjoint obstacles, or using convex fat but non-disjoint obstacles, or using fat disjoint obstacles that are non-convex. Our convex construction is easily generalizable to several other obstacle types, however, it always requires objects whose minimum bounding box has two long edges (polynomially long in $n, 1/\varepsilon$ and the spread) and one short edge of length 1.

Some notable obstacle types where the possible realizable metrics remain unexplored include disjoint axis-parallel box obstacles and disjoint $1 \times 1 \times n$ boxes of arbitrary orientation. We suspect that obstacles of the former type cannot realize all metric spaces, and obstacles of the latter type can only realize spaces of small doubling dimension. It also remains open whether exact realization of any metric space is possible with convex obstacles, even if all distances in the metric space are either 1 or 2.

Finally, can we generalize the planar studies of distances among weighted regions [9,17,30] to $\mathbb{R}^3$? What metrics can be realized if we are allowed to pass through the objects of $\mathcal{T}$, but we need to pay some custom penalty (weight) for doing so?

References

1 Mohammad Ali Abam, Marjan Adeli, Hamid Homapour, and Pooya Zafar Asadollahpoor. Geometric Spanners for Points Inside a Polygonal Domain. In Lars Arge and János Pach, editors, *31st International Symposium on Computational Geometry (SoCG 2015)*, volume 34 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 186–197, Dagstuhl, Germany, 2015. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SOCG.2015.186.

2 Mohammad Ali Abam and Mohammad Javad Rezaei Seraji. Geodesic spanners for points in R^3 amid axis-parallel boxes. *Information Processing Letters*, 166:106063, 2021. doi:10.1016/j.ipl.2020.106063.

3 Pankaj K. Agarwal, Dan Halperin, Micha Sharir, and Alex Steiger. Near-optimal min-sum motion planning for two square robots in a polygonal environment. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 4942–4962, 2024. doi:10.1137/1.9781611977912.176.

4 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.

5 Takao Asano, Tetsuo Asano, Leonidas Guibas, John Hershberger, and Hiroshi Imai. Visibility of disjoint polygons. *Algorithmica*, 1(1):49–63, November 1986. doi:10.1007/BF01840436.

6 Sandip Banerjee, Yair Bartal, Lee-Ad Gottlieb, and Alon Hovav. Novel Properties of Hierarchical Probabilistic Partitions and Their Algorithmic Applications . In *2024 IEEE 65th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1724–1767, Los Alamitos, CA, USA, 2024. IEEE Computer Society. doi:10.1109/FOCS61266.2024.00107.

7 Luis Barba, Michael Hoffmann, Matias Korman, and Alexander Pilz. Convex Hulls in Polygonal Domains. In David Eppstein, editor, *16th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2018)*, volume 101 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:13, Dagstuhl, Germany, 2018. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SWAT.2018.8.

8 Yair Bartal, Lee-Ad Gottlieb, and Robert Krauthgamer. The traveling salesman problem: Low-dimensionality implies a polynomial time approximation scheme. *SIAM J. Comput.*, 45(4):1563–1581, 2016. doi:10.1137/130913328.

9 Prosenjit Bose, Guillermo Esteban, and Anil Maheshwari. Weighted shortest path in equilateral triangular meshes. In Yeganeh Bahoo and Konstantinos Georgiou, editors, *Proceedings of the 34th Canadian Conference on Computational Geometry, CCCG 2022, Toronto Metropolitan University, Toronto, Ontario, Canada, August 25-27, 2022*, pages 60–67, 2022.

10 John F. Canny. *The complexity of robot motion planning*. MIT Press, Cambridge, MA, USA, 1988.

11 L.Paul Chew, Haggai David, Matthew J. Katz, and Klara Kedem. Walking around fat obstacles. *Information Processing Letters*, 83(3):135–140, 2002. doi:10.1016/S0020-0190(01)00321-0.

12 V. Chvátal. A combinatorial theorem in plane geometry. *Journal of Combinatorial Theory, Series B*, 18(1):39–41, February 1975. doi:10.1016/0095-8956(75)90061-1.

13 Kenneth L. Clarkson. Approximation algorithms for shortest path motion planning (extended abstract). In Alfred V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 56–65. ACM, 1987. doi:10.1145/28395.28402.

14 Sarita de Berg, Tillmann Miltzow, and Frank Staals. Towards Space Efficient Two-Point Shortest Path Queries in a Polygonal Domain. In Wolfgang Mulzer and Jeff M. Phillips, editors, *40th International Symposium on Computational Geometry (SoCG 2024)*, volume 293 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 17:1–17:16, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SoCG.2024.17.

15 Adrian Dumitrescu and Csaba D. Tóth. Maximal distortion of geodesic diameters in polygonal domains. In *Combinatorial Algorithms - 34th International Workshop, IWOCA 2023, Tainan, Taiwan, June 7-10, 2023, Proceedings*, volume 13889 of *Lecture Notes in Computer Science*, pages 197–208. Springer, 2023. doi:10.1007/978-3-031-34347-6_17.

16 Sándor P. Fekete, Stephan Friedrichs, Michael Hemmer, Joseph S. B. Mitchell, and Christiane Schmidt. On the chromatic art gallery problem. In *Canadian Conference on Computational Geometry (CCCG)*, 2014. URL: http://www.cccg.ca/proceedings/2014/papers/paper11.pdf.

17 L. Gewali, A. Meng, J. S. Mitchell, and S. Ntafos. Path planning in 0/1/ weighted regions with applications. In *Proceedings of the Fourth Annual Symposium on Computational Geometry*, SCG '88, pages 266–278, New York, NY, USA, 1988. Association for Computing Machinery. doi:10.1145/73393.73421.

18 Jacob E. Goodman and Joseph O'Rourke, editors. *Handbook of discrete and computational geometry*. CRC Press Series on Discrete Mathematics and its Applications. CRC Press, Boca Raton, FL, 1997.

19 D. Halperin, L. Kavraki, and K. Solovey. *Robotics*, chapter 51, pages 1343–1376. Chapman & Hall/CRC, 3rd edition, 2018.

20 D. Halperin, M. Sharir, and O. Salzman. *Algorithmic Motion Planning*, chapter 50, pages 1311–1342. Chapman & Hall/CRC, 3rd edition, 2018. doi:10.1201/9781315119601.

21 Sariel Har-Peled. Constructing approximate shortest path maps in three dimensions. *SIAM J. Comput.*, 28(4):1182–1197, 1999. doi:10.1137/S0097539797325223.

22 Juha Heinonen. *Lectures on Analysis on Metric Spaces*, volume 1 of *Universitext*. Springer New York, NY, 1 edition, 2001. doi:10.1007/978-1-4613-0131-8.

23 John Hershberger and Subhash Suri. An optimal algorithm for Euclidean shortest paths in the plane. *SIAM Journal on Computing*, 28(6):2215–2256, 1999. doi:10.1137/S0097539795289604.

24 J. E. Hopcroft, J. T. Schwartz, and M. Sharir. On the complexity of motion planning for multiple independent objects; PSPACE-hardness of the "warehouseman's problem". *International Journal of Robotics Research*, 3(4):76–88, 1984.

25 Marek Karpinski, Michael Lampis, and Richard Schmied. New inapproximability bounds for TSP. *Journal of Computer and System Sciences*, 81(8):1665–1677, 2015. doi:10.1016/j.jcss.2015.06.003.

26 Sándor Kisfaludi-Bak and Leonidas Theocharous. Realizing metric spaces with convex obstacles. arXiv:2509.14709, 2025. doi:10.48550/arXiv.2509.14709.

27 Anna Lubiw and Anurag Murty Naredla. The Visibility Center of a Simple Polygon. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *29th Annual European Symposium on Algorithms (ESA 2021)*, volume 204 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 65:1–65:14, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2021.65.

28 Joseph S. B. Mitchell. Shortest paths among obstacles in the plane. In *Proceedings of the Ninth Annual Symposium on Computational Geometry*, SCG '93, pages 308–317, New York, NY, USA, 1993. Association for Computing Machinery. doi:10.1145/160985.161156.

29 Joseph S. B. Mitchell. Approximating watchman routes. In *Proceedings of the 2013 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 844–855, 2013. doi:10.1137/1.9781611973105.60.

30 Joseph S. B. Mitchell and Christos H. Papadimitriou. The weighted region problem: finding shortest paths through a weighted planar subdivision. *J. ACM*, 38(1):18–73, January 1991. doi:10.1145/102782.102784.

31 Joseph S. B. Mitchell and Micha Sharir. New results on shortest paths in three dimensions. In *Proceedings of the 20th ACM Symposium on Computational Geometry, Brooklyn, New York, USA, June 8-11, 2004*, pages 124–133. ACM, 2004. doi:10.1145/997817.997839.

32 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.2019.04.003.

33 Joseph O'Rourke. *Art Gallery Theorems and Algorithms*. Oxford University Press, Inc., New York, NY, USA, 1987.

34 R. Pollack, M. Sharir, and G. Rote. Computing the geodesic center of a simple polygon. *Discrete & Computational Geometry*, 4(6):611–626, 1989. doi:10.1007/BF02187751.

35 Christian Rieck and Christian Scheffer. The Dispersive Art Gallery Problem. 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 67:1–67:18, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ISAAC.2022.67.

36 Jacob T Schwartz and Micha Sharir. On the "piano movers" problem. II. general techniques for computing topological properties of real algebraic manifolds. *Advances in Applied Mathematics*, 4(3):298–351, 1983. doi:10.1016/0196-8858(83)90014-3.

37 Jacob T. Schwartz and Micha Sharir. On the piano movers' problem: III. coordinating the motion of several independent bodies: The special case of circular bodies moving amidst polygonal barriers. *The International Journal of Robotics Research*, 2(3):46–75, 1983. doi:10.1177/027836498300200304.

38 Micha Sharir and Amir Schorr. On shortest paths in polyhedral spaces. In *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, STOC '84, pages 144–153, New York, NY, USA, 1984. Association for Computing Machinery. doi:10.1145/800057.808676.

39 G. T. Toussaint. An optimal algorithm for computing the relative convex hull of a set of points in a polygon. *EURASIP, Signal Processing III: Theories Applications, Part 2*, pages 853–856, 1986.

40 Haitao Wang. Shortest paths among obstacles in the plane revisited. In *Proceedings of the Thirty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '21, pages 810–821, USA, 2021. Society for Industrial and Applied Mathematics. doi:10.1137/1.9781611976465.51.

41 Glenn E. Wangdahl, Stephen M. Pollock, and John B. Woodward. Minimum-trajectory pipe routing. *Journal of Ship Research*, 18(01):46–49, March 1974. doi:10.5957/jsr.1974.18.1.46.

Hardness and Fixed Parameter Tractability for Pinwheel Scheduling Problems

Yusuke Kobayashi
Research Institute for Mathematical Sciences, Kyoto University, Japan

Bingkai Lin
School of Computer Science, Nanjing University, China

Abstract

In the Pinwheel Packing problem, we are given a set of recurring tasks, each associated with a positive integer a_i for task i. The objective is to select one task to perform each day such that every task i is performed at least once within every a_i consecutive days. The exact computational complexity of this problem, where $\sum 1/a_i = 1$, has remained an open question for more than 30 years; in particular, it is still unknown whether the problem is NP-hard. The first contribution of this paper is to show that Pinwheel Packing cannot be solved in polynomial time under a standard complexity assumption, improving upon the hardness result shown by Jacobs and Longo. Additionally, we present fixed-parameter algorithms for variants of Pinwheel Packing, parameterized by the number of tasks.

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

Keywords and phrases Pinwheel Scheduling, Polynomial-time Solvability, Packing and Covering, Fixed Parameter Algorithms

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.47

Funding *Yusuke Kobayashi*: JSPS KAKENHI Grant Numbers JP22H05001 and JP24K02901.

1 Introduction

1.1 Pinwheel Scheduling

The (packing version of) *Pinwheel Scheduling* problem was introduced by Holte et al. [12] in 1989 as a formalization of a real-time scheduling problem. In the problem, we are given k recurring tasks and a positive integer a_i for each task $i \in [k] = \{1, \dots, k\}$. The objective is to select one task to perform each day so that each task i is performed at least once every a_i days. In other words, we want to find pairwise disjoint sets $S_1, \dots, S_k \subseteq \mathbb{Z}$ such that

$$|[m, m + a_i) \cap S_i| \geq 1 \tag{1}$$

for all $i \in [k]$ and $m \in \mathbb{Z}$, where S_i represents the set of days on which task i is performed. Note that each S_i can be an infinite set. A natural computational problem is to determine the existence of such sets $S_1, \dots, S_k \subseteq \mathbb{Z}$, which is formally described as follows. Note that we call this problem Pinwheel Packing in this paper, because we will discuss the corresponding *covering* variant later.

Pinwheel Packing
Input: A sequence of positive integers $A = (a_i)_{i \in [k]}$.
Task: Determine whether there exist pairwise disjoint sets $S_1, \dots, S_k \subseteq \mathbb{Z}$ such that $\lvert[m, m + a_i) \cap S_i\rvert \geq 1$ for all $i \in [k]$ and $m \in \mathbb{Z}$.

© Yusuke Kobayashi and Bingkai Lin;
licensed under Creative Commons License CC-BY 4.0

Despite the simplicity and naturalness of the problem setting, little is known about the computational complexity of this problem. It has been shown in [12, Corollary 2.2] that Pinwheel Packing is in PSPACE; however, it is unknown whether this problem is in NP. Furthermore, it is unknown whether this problem is NP-hard. See [19] for details.

Since each task i has to be performed on at least a $\frac{1}{a_i}$ fraction of a sufficiently long consecutive sequence of days, if $A = (a_i)_{i\in[k]}$ is a yes-instance of Pinwheel Packing, then $\sum_{i=1}^{k} \frac{1}{a_i}$ (called the *density* of A) is at most 1. An instance $A = (a_i)_{i\in[k]}$ is called *dense* if its density is exactly 1. As we see in Section 1.2, for dense instances, Pinwheel Packing is equivalent to several other problems, highlighting that dense instances form an interesting special case. However, the exact complexity of dense instances of Pinwheel Packing has been open for more than 30 years. In fact, it is noted in [12, Section 4] that the NP-hardness of this problem remains an open question, and it is explicitly conjectured by Kawamura and Soejima [21, Conjecture 19] that Pinwheel Packing for dense instances is NP-complete. Note that the NP-hardness has been shown when the input sequence is represented in a concise notation; see [12, Theorem 4.12] and [19, Theorem 5]. Jacobs and Longo showed in their preprint [16] that (dense instances of) Pinwheel Packing cannot be solved in polynomial time unless SAT with n variables has a randomized algorithm running in expected time $n^{O(\log n \log\log n)}$.

1.2 Exact Packing and Covering

As mentioned in the previous subsection, for dense instances, Pinwheel Packing is equivalent to several other problems.

Observe that, if $S_1, \dots S_k \subseteq \mathbb{Z}$ form a solution for a dense instance $A = (a_i)_{i\in[k]}$ of Pinwheel Packing, then they satisfy (1) with equality. Using this observation, for dense instances, Pinwheel Packing is equivalent to the following *exact* variant.

Exact Pinwheel Packing
Input: A sequence of positive integers $A = (a_i)_{i\in[k]}$.
Task: Determine whether there exist pairwise disjoint sets $S_1, \dots, S_k \subseteq \mathbb{Z}$ such that $\lvert[m, m + a_i) \cap S_i\rvert = 1$ for all $i \in [k]$ and $m \in \mathbb{Z}$.

Note that the desired condition of this problem means that each task i is performed *exactly* once every a_i days. Therefore, S_i satisfies $\lvert[m, m + a_i) \cap S_i\rvert = 1$ for all $m \in \mathbb{Z}$ if and only if it is a *residue class* modulo a_i, i.e., $S_i = \{r_i + a_i \cdot q : q \in \mathbb{Z}\}$ for some integer r_i. Independently of its connection to Pinwheel Packing, Exact Pinwheel Packing is an interesting problem in its own right. Indeed, Exact Pinwheel Packing has been studied under the name *(1-server) periodic maintenance problem* [3, 25, 30] in the context of scheduling problems, and its solution has also garnered attention in discrete mathematics under the name *disjoint residue classes* [15, 21, 28]. It is shown in [3, 16, 21, 25] that Exact Pinwheel Packing is NP-complete, where we note that the input is not necessarily dense.

We can also consider a problem that is, in a sense, dual to (Exact) Pinwheel Packing, which is called (Exact) Pinwheel Covering [19]. Consider a scenario where a single task must be performed every day, and it is to be handled by k agents. Each agent $i \in [k]$ is assigned a positive integer a_i, which represents the condition that the agent can perform the task at most once (or exactly once) every a_i days. In this setting, the problem of determining the assignment of agents corresponds to finding a collection of sets $S_1, \dots, S_k \subseteq \mathbb{Z}$ such that $\bigcup_{i\in[k]} S_i = \mathbb{Z}$ and

$$\lvert[m, m + a_i) \cap S_i\rvert \leq 1 \quad (\text{or } \lvert[m, m + a_i) \cap S_i\rvert = 1)$$

for all $i \in [k]$ and $m \in \mathbb{Z}$, where S_i represents the set of days on which agent i performs the task. That is, the problem is formally described as follows.

Pinwheel Covering (resp. Exact Pinwheel Covering)
Input: A sequence of positive integers $A = (a_i)_{i\in[k]}$.
Task: Determine whether there exist sets $S_1, \ldots, S_k \subseteq \mathbb{Z}$ such that $\bigcup_{i\in[k]} S_i = \mathbb{Z}$ and $\lvert[m, m + a_i) \cap S_i\rvert \leq 1$ (resp. $\lvert[m, m + a_i) \cap S_i\rvert = 1$) for all $i \in [k]$ and $m \in \mathbb{Z}$.

Pinwheel Covering was introduced and studied under the name *point patrolling* in [21]. Similarly to the packing problem, this problem is in PSPACE, and it is unknown whether this problem is NP-hard. In contrast, very recently, it was shown that Exact Pinwheel Covering is NP-hard [20]. Note that feasible solutions for Exact Pinwheel Covering are known as *Erdős covering systems* and have been actively studied by number theorists; see [2, 14].

Since each agent i can perform a task on at most a $\frac{1}{a_i}$ fraction of a sufficiently long consecutive sequence of days, if $A = (a_i)_{i\in[k]}$ is a yes-instance of Pinwheel Covering, then its density $\sum_{i=1}^{k} \frac{1}{a_i}$ is at least 1. Similarly to the packing problem, if the density is exactly 1, then Pinwheel Covering and Exact Pinwheel Covering are equivalent. Furthermore, if the density of $A = (a_i)_{i\in[k]}$ is exactly 1 and $S_1, \ldots, S_k \subseteq \mathbb{Z}$ satisfy $\lvert[m, m + a_i) \cap S_i\rvert = 1$ for all $i \in [k]$ and $m \in \mathbb{Z}$, then we see that $\bigcup_{i\in[k]} S_i = \mathbb{Z}$ if and only if $S_1, \ldots, S_k$ are pairwise disjoint. By combining these facts, we arrive at the following observation.

▶ **Observation 1** (see [19, Section 5]). *For instances with density exactly* 1*, the four above problems, i.e.,* (Exact) Pinwheel Packing *and* (Exact) Pinwheel Covering*, coincide.*

For brevity, Pinwheel Packing, when restricted to dense instances, is referred to as Dense Pinwheel Packing. Then, Observation 1 implies that Dense Pinwheel Packing is a special case of all the four above problems.

1.3 Our Contributions

As described in Section 1.1, it was shown by Jacobs and Longo [16] that Dense Pinwheel Packing cannot be solved in polynomial time unless SAT with n variables has a randomized algorithm running in expected time $n^{O(\log n \log\log n)}$. The first contribution of this paper is to strengthen this result by showing that Dense Pinwheel Packing cannot be solved in polynomial time under a weaker complexity assumption, partially addressing the open question in [12, Section 4] and [21, Conjecture 19].

▶ **Theorem 2.** Dense Pinwheel Packing *cannot be solved in polynomial time unless* $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{O(\log n)})$.

Here, $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{O(\log n)})$ means that every problem in NP can be solved in $n^{O(\log n)}$ time, where n is the input size of the problem. The assumption $\mathsf{NP} \not\subseteq \mathsf{DTIME}(n^{O(\log n)})$ is weaker than the assumption in [16] in two key aspects: the running time is improved from $n^{O(\log n \log\log n)}$ to $n^{O(\log n)}$, and our assumption does not rely on randomization. Note that the assumption $\mathsf{NP} \not\subseteq \mathsf{DTIME}(n^{O(\log n)})$ is weaker than the Exponential Time Hypothesis (ETH) and $\mathsf{NP} \not\subseteq \mathsf{QP}$ (see Corollary 10), while it is stronger than $\mathsf{P} \neq \mathsf{NP}$. Theorem 2 implies that Pinwheel Packing and Pinwheel Covering cannot be solved in polynomial time under the same assumption.

▶ **Corollary 3.** *Neither* Pinwheel Packing *nor* Pinwheel Covering *can be solved in polynomial time unless* $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{O(\log n)})$.

In our proof for Theorem 2, we refine the argument in [16]. We first prove that Exact Pinwheel Packing is hard even when $\mathrm{lcm}(A)$ is not particularly large (see Theorem 6), where $\mathrm{lcm}(A)$ denotes the least common multiple of $a_1, \ldots, a_k$ for $A = (a_i)_{i \in [k]}$. We then reduce Exact Pinwheel Packing to Dense Pinwheel Packing by appending multiple copies of integer $\mathrm{lcm}(A)$, thereby proving the hardness of Dense Pinwheel Packing. The primary technical challenge lies in ensuring that $\mathrm{lcm}(A)$ does not become excessively large.

To achieve this, Jacobs and Longo [16] reduce an NP-hard problem of size n to a new problem called Partial Coding, and then further reduce it to Exact Pinwheel Packing. These reductions show that Exact Pinwheel Packing is hard even when $\mathrm{lcm}(A) = n^{O(\log n \log \log n)}$. In contrast, we directly and carefully reduce an NP-hard problem to Exact Pinwheel Packing, which enables us to achieve a tighter bound of $\mathrm{lcm}(A) = n^{O(\log n)}$. To derandomize our reduction step, we utilize a sophisticated tool known as (n, k)-*universal sets* [26] in a non-trivial manner, which constitutes the most technical aspect of our proof. For details, see Section 2.

Theorem 2 suggests that (Exact) Pinwheel Packing and (Exact) Pinwheel Covering are unlikely to be solvable in polynomial time. It is then natural to consider the fixed-parameter tractability of these problems. Indeed, Gąsieniec et al. [10] showed that Pinwheel Packing can be solved in FPT time when parameterized by k. Our second contribution is to show that the remaining three variants can also be solved in FPT time with respect to the parameter k.

▶ **Theorem 4.** Pinwheel Covering *and* Exact Pinwheel Covering *can be solved in* $O(n) + k^{O(2^k)}$ *time, where* $n := \sum_{i=1}^{k} \log a_i$ *is the input size of the problem.*

▶ **Theorem 5.** Exact Pinwheel Packing *can be solved in* $(\log k)^{O(k^2)} \cdot n^{O(1)}$ *time, where* $n := \sum_{i=1}^{k} \log a_i$ *is the input size of the problem.*

We here describe proof ideas for these theorems, while formal proofs are given in Section 3. Our proof for Theorem 4 is based on the following intuition: if some a_i is extremely large, then removing a_i does not affect the feasibility of the problems. Our technical contribution lies in formalizing this intuition. We then show that it is sufficient to consider the case where each a_i is bounded by a function of k, leading to FPT algorithms.

It should be noted that this argument does not apply to Exact Pinwheel Packing, as removing a large a_i could affect the feasibility of the problem. For instance, if A consists of two large integers a_1 and a_2 that are coprime, then A is a no-instance. However, removing either a_1 or a_2 would result in a yes-instance. Therefore, we adopt a completely different approach to obtain Theorem 5. We observe that a solution to Exact Pinwheel Packing consists of sets $S_i = \{r_i + a_i \cdot q : q \in \mathbb{Z}\}$, where each set is determined by an integer r_i. Treating $r_1, \ldots, r_k$ as integer variables, we show that the constraints for r_i can be formulated as an integer linear program by introducing several auxiliary variables. Since the integer linear programming problem can be solved in FPT time, parameterized by the number of variables, this yields an FPT algorithm for Exact Pinwheel Packing.

Moreover, to highlight the similarities and differences between Pinwheel Packing and (Exact) Pinwheel Covering, an FPT algorithm for Pinwheel Packing with an explicit running time bound is provided in Appendix A.

1.4 Related Work

A generalization of Pinwheel Packing has been studied under the name of Windows Scheduling Problem. While Pinwheel Packing can be viewed as a scheduling problem involving a single machine where all tasks have the same size, Windows Scheduling Problem extends this to a setting with multiple machines and tasks of varying sizes [4, 5, 16, 22].

Bamboo Garden Trimming problem is an optimization variant of Pinwheel Packing. Here, the objective is to find the smallest positive number h such that $(h \cdot a_i)_{i\in[k]}$ is a yes-instance of Pinwheel Packing. Hardness results and approximation algorithms have been investigated for this problem [9, 11, 29].

For Pinwheel Packing, there has been extensive research on sufficient conditions for the density of the input A to be a yes-instance. It is not so difficult to show that A is a yes-instance if its density is at most $\frac{1}{2}$ [12]. A central question in this area is whether the same statement holds when $\frac{1}{2}$ is replaced with a larger number α. The maximum such number α is called the *density threshold*. Since $A = (2, 3, m)$ is a no-instance with a density of $\frac{5}{6} + \frac{1}{m}$ for any integer m, the density threshold is at most $\frac{5}{6}$. Building on extensive research on the density threshold [6, 7, 10, 24], Kawamura [18] recently showed that the density threshold is exactly $\frac{5}{6}$, resolving the conjecture proposed by Chan and Chin [6]. A threshold depending on the smallest element in A is investigated in [11]. It is also shown that if A consists of at most two distinct values and its density is at most 1, then it is a yes-instance [13].

2 Hardness Results

The objective of this section is to show the hardness of Dense Pinwheel Packing (Theorem 2). To achieve this, we reduce a well-known NP-hard problem called 3-Coloring in Graphs with Maximum Degree Four; see [8]. In 3-Coloring in Graphs with Maximum Degree Four, the input is a graph $G = (V, E)$ with maximum degree four and the objective is to decide whether G is 3-colorable, that is, to decide whether we can color V with three colors so that adjacent vertices have different colors.

We first show in Section 2.1 that 3-Coloring in Graphs with Maximum Degree Four with n vertices can be reduced to Exact Pinwheel Packing with $\mathrm{lcm}(A) = n^{O(\log n)}$. Recall that $\mathrm{lcm}(A)$ denotes the least common multiple of $a_1, \ldots, a_k$ for $A = (a_i)_{i\in[k]}$. Then, in Section 2.2, we reduce Exact Pinwheel Packing with $\mathrm{lcm}(A) = n^{O(\log n)}$ to Dense Pinwheel Packing of size $n^{O(\log n)}$.

2.1 Reduction to Exact Pinwheel Packing

▶ **Theorem 6.** *Let $G = (V, E)$ be a graph with $|V| = n$ whose maximum degree is four. In $n^{O(1)}$ time, we can construct a sequence of positive integers A with $\mathrm{lcm}(A) = n^{O(\log n)}$ such that $G = (V, E)$ is 3-colorable if and only if A is a yes-instance of* Exact Pinwheel Packing.

Proof. We give a reduction from 3-Coloring in Graphs with Maximum Degree Four to Exact Pinwheel Packing. It suffices to consider the case where n is sufficiently large.

Reduction. Let $G = (V, E)$ be a graph with $|V| = n$ whose maximum degree is four. We first construct a set T and a family $(T_v)_{v\in V}$ of subsets of T by applying the following technical lemma, whose proof is postponed to Section 2.3.

▶ **Lemma 7.** *Let $G = (V, E)$ be a graph with $|V| = n$ whose maximum degree is four. In $n^{O(1)}$ time, we can construct a set T with $|T| = O(\log n)$ and its subsets $T_v \subseteq T$ each indexed by $v \in V$ such that vertices v and v' are adjacent in G if and only if $T_v \cap T_{v'} = \emptyset$.*

We pick $|T|$ distinct primes $(p_i)_{i \in T}$ indexed by T such that $n < p_i < 2n$ for each $i \in T$. Note that such primes exist when n is sufficiently large, because the prime number theorem guarantees that, for any positive integer N, there are $\Theta\left(\frac{N}{\log N}\right)$ primes less than or equal to N. For every $v \in V$, let

$$a_v = 3 \cdot \prod_{i \in T_v} p_i.$$

Then, we obtain an instance $A = (a_v)_{v \in V}$ of Exact Pinwheel Packing. The construction can be done in $n^{O(1)}$-time, and

$$\operatorname{lcm}(A) \leq 3 \cdot \left(\max_{i \in T} p_i\right)^{|T|} = n^{O(\log n)}.$$

Completeness. We show that if G is 3-colorable, then $A = (a_v)_{v \in V}$ is a yes-instance of Exact Pinwheel Packing. Suppose $G = (V, E)$ has a 3-coloring $c\colon V \to [3]$. Let ℓ be a bijection from V to $[n]$. For each $v \in V$, let r_v be an integer such that

$$r_v \equiv c(v) \mod 3, \tag{2}$$

and

$$r_v \equiv \ell(v) \mod \prod_{i \in T_v} p_i, \tag{3}$$

whose existence is guaranteed by the Chinese remainder theorem. Let $S_v = \{r_v + a_v \cdot q : q \in \mathbb{Z}\}$ for $v \in V$. We claim that $(S_v)_{v \in V}$ is a solution to the Exact Pinwheel Packing instance $A = (a_v)_{v \in V}$, that is, $S_v \cap S_{v'} = \emptyset$ holds for any pair of distinct $v, v' \in V$. We consider the following two cases separately.

- Suppose that v and v' are adjacent in G. Then, we have $c(v) \neq c(v')$ as c is a coloring, and hence $r_v \not\equiv r_{v'} \pmod 3$ by (2). Thus, for all $q, q' \in \mathbb{Z}$,

  $$r_v + a_v \cdot q \neq r_{v'} + a_{v'} \cdot q',$$

  because both a_v and $a_{v'}$ are divisible by 3. This shows that $S_v \cap S_{v'} = \emptyset$.
- Suppose that v and v' are not adjacent in G. Then, $T_v \cap T_{v'} \neq \emptyset$ holds by the construction in Lemma 7. Let i be an element in $T_v \cap T_{v'}$. Since $p_i > n > |\ell(v) - \ell(v')| \geq 1$, we have $\ell(v) \not\equiv \ell(v') \pmod{p_i}$, which implies that

  $$r_v \equiv \ell(v) \not\equiv \ell(v') \equiv r_{v'} \mod p_i$$

  by (3). Thus, for all $q, q' \in \mathbb{Z}$,

  $$r_v + a_v \cdot q \neq r_{v'} + a_{v'} \cdot q',$$

  because both a_v and $a_{v'}$ are divisible by p_i. This shows that $S_v \cap S_{v'} = \emptyset$.

Soundness. We show that if $A = (a_v)_{v \in V}$ is a yes-instance of Exact Pinwheel Packing, then G is 3-colorable. Suppose that $(S_v)_{v \in V}$ is a solution to the Exact Pinwheel Packing instance $A = (a_v)_{v \in V}$. For each $v \in V$, since S_v is a residue class modulo a_i, there exists an integer r_v such that $S_v = \{r_v + a_v \cdot q : q \in \mathbb{Z}\}$. For $v \in V$, let $c(v)$ be the integer in $\{1, 2, 3\}$ such that $c(v) \equiv r_v \pmod 3$. This defines a function $c \colon V \to [3]$. We claim that c is a coloring of G. Let $v, v' \in V$ be vertices that are adjacent to each other. Then, we have $T_v \cap T_{v'} = \emptyset$, and hence $\gcd(a_v, a_{v'}) = 3$, where gcd denotes the greatest common divisor. Since $S_v \cap S_{v'} = \emptyset$ is equivalent to $r_v \not\equiv r_{v'} \pmod{\gcd(a_v, a_{v'})}$ by the Chinese remainder theorem (see e.g. [3, Lemma 12]), we obtain

$$c(v) \equiv r_v \not\equiv r_{v'} \equiv c(v') \mod 3.$$

Therefore, c is a 3-coloring of G. ◀

▶ Remark 8. Theorem 6 gives an alternative proof of the NP-completeness of Exact Pinwheel Packing, which has already been shown in [3, 16, 21, 25]. The condition $\mathrm{lcm}(A) = n^{O(\log n)}$ is not required to prove the NP-completeness of Exact Pinwheel Packing. However, it is used to show the hardness of Dense Pinwheel Packing in the next subsection.

2.2 Hardness of Dense Pinwheel Packing

In this subsection, we prove the hardness of Dense Pinwheel Packing using Theorem 6.

▶ **Theorem 9.** *Let $G = (V, E)$ be a graph with $|V| = n$ whose maximum degree is four. In $n^{O(\log n)}$ time, we can construct an instance $\hat{A} = (a_i)_{i \in [k]}$ of* Dense Pinwheel Packing *such that $k = n^{O(\log n)}$, $a_i = n^{O(\log n)}$ for each $i \in [k]$, and $G = (V, E)$ is 3-colorable if and only if $\hat{A}$ is a yes-instance of* Dense Pinwheel Packing.

Proof. Let $G = (V, E)$ be a graph with $|V| = n$ whose maximum degree is four. By Theorem 6, in $n^{O(1)}$ time, we construct a sequence of positive integers $A = (a_i)_{i \in [\ell]}$ with $\mathrm{lcm}(A) = n^{O(\log n)}$ such that $G = (V, E)$ is 3-colorable if and only if A is a yes-instance of Exact Pinwheel Packing. We may assume that the density of A is at most 1, since otherwise we can just return a trivial no-instance.

We construct an instance $\hat{A}$ of Dense Pinwheel Packing from A by appending multiple copies of integer $\mathrm{lcm}(A)$ so that the density becomes 1. That is, we define $\hat{A} = (a_i)_{i \in [k]}$ by

$$k = \ell + \left(1 - \sum_{i=1}^{\ell} \frac{1}{a_i}\right) \mathrm{lcm}(A),$$

$$a_{\ell+1} = a_{\ell+2} = \cdots = a_k = \mathrm{lcm}(A).$$

Then, we see that $k \leq \mathrm{lcm}(A) = n^{O(\log n)}$ and $a_i \leq \mathrm{lcm}(A) = n^{O(\log n)}$ for each $i \in [k]$. This implies that $\hat{A}$ can be constructed in $n^{O(\log n)}$ time.

It remains to show that $G = (V, E)$ is 3-colorable if and only if $\hat{A}$ is a yes-instance of Dense Pinwheel Packing. To this end, we prove that A is a yes-instance of Exact Pinwheel Packing if and only if $\hat{A}$ is a yes-instance of Dense Pinwheel Packing.

Suppose first that $\hat{A}$ is a yes-instance of Dense Pinwheel Packing and let $(S_i)_{i \in [k]}$ be a solution for it. Then, $(S_i)_{i \in [\ell]}$ is a solution to the Exact Pinwheel Packing instance A.

Suppose next that A is a yes-instance of Exact Pinwheel Packing and let $(S_i)_{i \in [\ell]}$ be a solution for it. Since each S_i is a residue class modulo a_i, it is the union of $\frac{\mathrm{lcm}(A)}{a_i}$ residue classes modulo $\mathrm{lcm}(A)$. Then, $\mathbb{Z} \setminus \bigcup_{i=1}^{\ell} S_i$ is the union of $\mathrm{lcm}(A) - \sum_{i=1}^{\ell} \frac{\mathrm{lcm}(A)}{a_i} = k - \ell$ residue classes modulo $\mathrm{lcm}(A)$. Therefore, we can take residue classes $S_{\ell+1}, \ldots, S_k$ modulo $\mathrm{lcm}(A)$ so that $(S_i)_{i \in [k]}$ is a solution to the Dense Pinwheel Packing instance $\hat{A}$. ◀

We are now ready to prove Theorem 2, which we restate here.

▶ **Theorem 2.** Dense Pinwheel Packing *cannot be solved in polynomial time unless* $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{O(\log n)})$.

Proof. We prove $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{O(\log n)})$ assuming that Dense Pinwheel Packing can be solved in polynomial time. Suppose we are given an instance of a problem in NP whose input size is n. Then, we can construct an equivalent instance of 3-Coloring in Graphs with Maximum Degree Four whose size is $n^{O(1)}$, because 3-Coloring in Graphs with Maximum Degree Four is NP-hard (see [8]). By Theorem 9, we construct an equivalent instance of Dense Pinwheel Packing whose size is $n^{O(\log n)}$. These reductions show that, if Dense Pinwheel Packing can be solved in polynomial time, then the original problem in NP can be solved in $n^{O(\log n)}$ time. Therefore, $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{O(\log n)})$ holds under the assumption that Dense Pinwheel Packing can be solved in polynomial time. ◀

Using the same reduction, we can also show that Dense Pinwheel Packing cannot be solved in quasi-polynomial time (i.e., $2^{\mathrm{polylog}(n)}$ time) under a stronger complexity assumption. Note that the same hardness result holds also for Pinwheel Packing and Pinwheel Covering, because Dense Pinwheel Packing is a special case of these problems.

▶ **Corollary 10.** Dense Pinwheel Packing *cannot be solved in quasi-polynomial time unless* $\mathsf{NP} \subseteq \mathsf{QP}$*, where* QP *is the class of problems that can be solved in quasi-polynomial time, i.e.,* $\mathsf{QP} = \mathsf{DTIME}(2^{\mathrm{polylog}(n)})$.

Proof. As in the proof of Theorem 2, any problem in NP with input size n can be reduced to Dense Pinwheel Packing with size $N = n^{O(\log n)}$. If Dense Pinwheel Packing can be solved in quasi-polynomial time, then the original problem can be solved in $2^{\mathrm{polylog}(N)} = 2^{\mathrm{polylog}(n)}$ time. Therefore, $\mathsf{NP} \subseteq \mathsf{QP}$ holds under the assumption that Dense Pinwheel Packing can be solved in quasi-polynomial time. ◀

2.3 Proof of Lemma 7

In this subsection, we give a proof of Lemma 7. Let $G = (V, E)$ be a graph with $|V| = n$ whose maximum degree is four. A vertex subset $I \subseteq V$ is called an *independent set* if no two vertices in I are adjacent to each other. Our idea is to find a family $\mathcal{I}$ of independent sets in G such that

(I) for any pair of non-adjacent vertices $v, v' \in V$, there exists $I \in \mathcal{I}$ such that $\{v, v'\} \subseteq I$.

▷ Claim 11. Suppose we are given an independent set family $\mathcal{I}$ of size $O(\log n)$ with property (I). Then, in polynomial time, we can construct a set T and a subset $T_v \subseteq T$ for each $v \in V$ satisfying the conditions in Lemma 7.

Proof of Claim 11. Let $\mathcal{I}$ be an independent set family of size $O(\log n)$ with property (I). Then, we can simply set $T = \mathcal{I}$ and let $T_v = \{I \in \mathcal{I} : v \in I\}$ for $v \in V$. We easily see that such T and $(T_v)_{v \in V}$ satisfy the conditions in Lemma 7 as follows.

- It is obvious that $|T| = |\mathcal{I}| = O(\log n)$.
- If vertices v and v' are adjacent, then they are not contained in the same independent set in G, and hence $T_v \cap T_{v'} = \emptyset$.
- If vertices v and v' are not adjacent, then there must exist an independent set $I \in \mathcal{I}$ such that $\{v, v'\} \subseteq I$ by property (I), and hence $I \in T_v \cap T_{v'}$.

This completes the proof of the claim. ◁

To find an independent set family $\mathcal{I}$ of size $O(\log n)$ that satisfies property (I), we use an argument inspired by the color-coding technique [1]. For every $v \in V$, let $N[v]$ denote its closed neighborhood, i.e., $N[v] = \{u \in V : vu \in E\} \cup \{v\}$. Observe that $|N[v] \cup N[v']| \leq 10$ for any distinct vertices $v, v' \in V$ as G is a graph with maximum degree four. We show that it suffices to construct a family $\mathcal{U} \subseteq 2^V$ of vertex subsets with $|\mathcal{U}| = O(\log n)$ such that

(U) for any pair of distinct vertices $v, v' \in V$, there exists a vertex subset $U \in \mathcal{U}$ such that $U \cap (N[v] \cup N[v']) = \{v, v'\}$.

▷ Claim 12. Suppose we are given a family $\mathcal{U} \subseteq 2^V$ of vertex subsets that satisfies property (U). Then, in polynomial time, we can construct an independent set family $\mathcal{I}$ of size at most $|\mathcal{U}|$ with property (I).

Proof of Claim 12. For each $U \in \mathcal{U}$, let I_U be an inclusionwise maximal independent set in G subject to $I_U \subseteq U$. Then, set $\mathcal{I} = \{I_U : U \in \mathcal{U}\}$, which can be computed in polynomial time. It is obvious that $|\mathcal{I}| \leq |\mathcal{U}|$.

Let $v, v' \in V$ be a pair of distinct non-adjacent vertices. By property (U), there exists a vertex subset $U \in \mathcal{U}$ such that $U \cap (N[v] \cup N[v']) = \{v, v'\}$. Since any maximal independent set contained in U contains both v and v', we obtain $\{v, v'\} \subseteq I_U$. This shows that $\mathcal{I}$ satisfies property (I). ◁

By Claims 11 and 12, to prove Lemma 7, it remains to construct in polynomial time a family $\mathcal{U} \subseteq 2^V$ with $|\mathcal{U}| = O(\log n)$ that satisfies property (U). Below, we describe a simple probabilistic construction of $\mathcal{U}$, followed by a deterministic construction using a sophisticated tool known as the (n, k)-*universal sets*.

Probabilistic Construction of $\mathcal{U}$

Consider a random vertex subset $U \subseteq V$ that contains each vertex independently with probability $\frac{1}{2}$. Then, for any pair of distinct vertices $v, v' \in V$, $U \cap (N[v] \cup N[v']) = \{v, v'\}$ holds with probability

$$\frac{1}{2^{|N[v] \cup N[v']|}} \geq \frac{1}{2^{10}}.$$

We pick such a random vertex subset U independently many times to obtain a family $\mathcal{U}$. Then, for a pair of distinct vertices $v, v' \in V$, $U \cap (N[v] \cup N[v']) \neq \{v, v'\}$ holds for every $U \in \mathcal{U}$ with probability at most

$$\left(1 - \frac{1}{2^{10}}\right)^{|\mathcal{U}|}.$$

By the union bound, with probability at least $1 - |V|^2 \left(1 - \frac{1}{2^{10}}\right)^{|\mathcal{U}|}$, $\mathcal{U}$ satisfies property (U). When $|\mathcal{U}| = \Theta(\log |V|) = \Theta(\log n)$, we can generate such a family $\mathcal{U}$ with constant probability in polynomial time.

Deterministic Construction of $\mathcal{U}$

We use the (n, k)-*universal sets* to construct a desired family $\mathcal{U}$ deterministically.

▶ **Definition 13** ((n, k)-universal set). *For positive integers n and k, an (n, k)-universal set is a set of vectors $W \subseteq \{0, 1\}^n$ such that for any index set $S \subseteq [n]$ of size k, the projection of W on S contains all 2^k possible configurations.*

▶ **Theorem 14** ([26, Theorem 2]). *For any positive integers n and k with $k \leq n$, an (n, k)-universal set of cardinality $O(k2^k \log n)$ can be constructed deterministically in time $O(\binom{n}{k} k 2^{2k} n^{\lfloor k/2 \rfloor})$.*

Setting $n = |V|$ and $k = 10$, by Theorem 14, we construct an (n, k)-universal set W of size $O(k2^k \log n) = O(\log n)$ in time $O(\binom{n}{k} k 2^{2k} n^{\lfloor k/2 \rfloor}) = n^{O(1)}$. Using an arbitrary bijection from V to $[n]$, $W \subseteq \{0,1\}^n$ can be regarded as a subset of $\{0,1\}^V$. By identifying a subset of V and its characteristic vector in $\{0,1\}^V$, W defines a family $\mathcal{U} \subseteq 2^V$. Then, $|\mathcal{U}| = |W| = O(\log n)$.

We see that $\mathcal{U}$ satisfies property (U) as follows. Let $v, v' \in V$ be distinct non-adjacent vertices. Since W is a $(|V|, 10)$-universal set and $|N[v] \cup N[v']| \leq 10$, there exists a vector $w \in W$ whose projection on $N[v] \cup N[v']$ is the characteristic vector of $\{v, v'\}$. This means that the set $U \in \mathcal{U}$ corresponding to w satisfies $U \cap (N[v] \cup N[v']) = \{v, v'\}$.

3 Fixed Parameter Algorithms

In this section, we present fixed parameter algorithms for (Exact) Pinwheel Covering and Exact Pinwheel Packing. We prove Theorems 4 and 5 in Sections 3.1 and 3.2, respectively.

3.1 Pinwheel Covering and Exact Pinwheel Covering

In this subsection, we deal with Pinwheel Covering and Exact Pinwheel Covering simultaneously and prove Theorem 4. We first present the following claim. Although this claim was already shown in [21, Section 4.1] for Pinwheel Covering, we here give a proof for completeness.

▷ Claim 15 ([21, Section 4.1]). Pinwheel Covering and Exact Pinwheel Covering can be solved in $\left(\prod_{i=1}^{k} a_i\right)^{O(1)}$ time.

Proof of Claim 15. We first consider Pinwheel Covering. Construct a digraph G with $\prod_{i=1}^{k} a_i$ vertices, each of which corresponds to a k-tuple $(b_1, \ldots, b_k)$ with $b_i \in \{0, 1, \ldots, a_i - 1\}$ for each $i \in [k]$. This vertex corresponds to a state in which agent i must wait for b_i days before handling the task again. We add an arc from a vertex $(b_1, \ldots, b_k)$ to a vertex $(b'_1, \ldots, b'_k)$ when there is $i \in [k]$ such that $b_i = 0$, $b'_i = a_i - 1$, and $b'_j = \max\{0, b_j - 1\}$ for $j \in [k] \setminus \{i\}$. Since a feasible solution of Pinwheel Covering corresponds to an infinite walk in G, A is a yes-instance if and only if G contains a dicycle. This can be checked in $\left(\prod_{i=1}^{k} a_i\right)^{O(1)}$ time.

To deal with Exact Pinwheel Covering, we modify the definition of the edge set as follows: we add an arc from a vertex $(b_1, \ldots, b_k)$ to a vertex $(b'_1, \ldots, b'_k)$ when there is $i \in [k]$ such that $b_i = 0$, $b'_i = a_i - 1$, and $(b_j, b'_j) = (0, a_j - 1)$ or $b'_j = b_j - 1$ for $j \in [k] \setminus \{i\}$.

Then, A is a yes-instance if and only if the digraph contains a dicycle. This can be checked in $\left(\prod_{i=1}^{k} a_i\right)^{O(1)}$ time. ◁

By this claim, to obtain an FPT algorithm for (Exact) Pinwheel Covering, it is sufficient to bound each a_i by a function of k. To achieve this, we prove the following two claims.

▷ Claim 16. Suppose that $A = (a_i)_{i \in [k]}$ is a no-instance of (Exact) Pinwheel Covering and $(S_i)_{i \in [k]}$ is a family of subsets of $\mathbb{Z}$ such that

$$|[m, m + a_i) \cap S_i| \leq 1 \quad (\text{or } |[m, m + a_i) \cap S_i| = 1)$$

for all $i \in [k]$ and $m \in \mathbb{Z}$. Then, no $\prod_{i=1}^{k} a_i$ consecutive integers are covered by $\bigcup_{i=1}^{k} S_i$, that is,

$$\left| \left[m, m + \prod_{i=1}^{k} a_i \right) \cap \left(\mathbb{Z} - \bigcup_{i=1}^{k} S_i \right) \right| \geq 1 \tag{4}$$

for all $m \in \mathbb{Z}$.

Proof of Claim 16. Suppose that $A = (a_i)_{i\in[k]}$ is a no-instance of (Exact) Pinwheel Covering. Then, the digraph in the proof of Claim 15 contains no dicycle. Therefore, the digraph contains no walk of length $\prod_{i=1}^{k} a_i$, which means that no $\prod_{i=1}^{k} a_i$ consecutive integers are covered by $\bigcup_{i=1}^{k} S_i$. ◁

▷ Claim 17. Let $A = (a_i)_{i\in[k]}$ be a sequence of positive integers with $a_1 \leq a_2 \leq \cdots \leq a_k$, and let $\ell \in [k-1]$. If A is a yes-instance of (Exact) Pinwheel Covering and $A' = (a_i)_{i\in[\ell]}$ is a no-instance of (Exact) Pinwheel Covering, then $a_{\ell+1} \leq (k-\ell) \prod_{i=1}^{\ell} a_i$.

Proof of Claim 17. Let $(S_i)_{i\in[k]}$ be a feasible solution for the instance A. Since $A' = (a_i)_{i\in[\ell]}$ is a no-instance of (Exact) Pinwheel Covering, by Claim 16, no $\prod_{i=1}^{\ell} a_i$ consecutive integers are covered by $\bigcup_{i=1}^{\ell} S_i$. Therefore, in any $\prod_{i=1}^{\ell} a_i$ consecutive integers, at least one integer is covered by $S_{\ell+1} \cup \cdots \cup S_k$. This implies that

$$\frac{1}{\prod_{i=1}^{\ell} a_i} \leq \frac{1}{a_{\ell+1}} + \cdots + \frac{1}{a_k} \leq \frac{k-\ell}{a_{\ell+1}},$$

and hence $a_{\ell+1} \leq (k-\ell) \prod_{i=1}^{\ell} a_i$. ◁

We are now ready to prove Theorem 4, which we restate here.

▶ **Theorem 4.** Pinwheel Covering *and* Exact Pinwheel Covering *can be solved in* $O(n) + k^{O(2^k)}$ *time, where* $n := \sum_{i=1}^{k} \log a_i$ *is the input size of the problem.*

Proof. Let $A = (a_i)_{i\in[k]}$ be an instance of (Exact) Pinwheel Covering. Without loss of generality, we may assume that $a_1 \leq a_2 \leq \cdots \leq a_k$. It suffices to consider the case where $a_1 \leq k$, since otherwise we can immediately conclude that A is a no-instance.

If $a_i \leq k^{2^{i-1}}$ for every $i \in [k]$, then we can check the feasibility of A in $\left(\prod_{i=1}^{k} a_i\right)^{O(1)} = k^{O(2^k)}$ time by Claim 15. Otherwise, let $\ell \in [k-1]$ be the minimum integer with $a_{\ell+1} > k^{2^\ell}$, which can be computed in $O(n)$ time. Then, we obtain

$$a_{\ell+1} > k^{2^\ell} = k \prod_{i=1}^{\ell} k^{2^{i-1}} > (k-\ell) \prod_{i=1}^{\ell} k^{2^{i-1}} > (k-\ell) \prod_{i=1}^{\ell} a_i, \tag{5}$$

where the last inequality is by the minimality of ℓ and $a_1 < k^{2^{1-1}}$. We observe that $(a_i)_{i\in[k]}$ is a yes-instance if and only if $(a_i)_{i\in[\ell]}$ is a yes-instance, as the "if" part is trivial and the "only if" part follows from Claim 17 and (5). We can check whether $(a_i)_{i\in[\ell]}$ is a yes-instance or not in $\left(\prod_{i=1}^{\ell} a_i\right)^{O(1)}$ time by Claim 15, and this running time is bounded by $k^{O(2^k)}$. ◀

3.2 Exact Pinwheel Packing

In this subsection, we deal with Exact Pinwheel Packing and prove Theorem 5.

▶ **Theorem 5.** Exact Pinwheel Packing *can be solved in* $(\log k)^{O(k^2)} \cdot n^{O(1)}$ *time, where* $n := \sum_{i=1}^{k} \log a_i$ *is the input size of the problem.*

Proof. Let $A = (a_i)_{i \in [k]}$ be an instance of Exact Pinwheel Packing. Recall that the objective is to find pairwise disjoint sets $S_1, \ldots, S_k$ such that S_i is a residue class modulo a_i for $i \in [k]$. This is equivalent to finding $r_i \in \mathbb{Z}$ for $i \in [k]$ such that

$$S_i = \{r_i + a_i \cdot q : q \in \mathbb{Z}\} \text{ and } S_j = \{r_j + a_j \cdot q : q \in \mathbb{Z}\} \text{ are disjoint} \tag{6}$$

for all distinct $i, j \in [k]$. By the Chinese remainder theorem (see e.g. [3, Lemma 12]), (6) is equivalent to

$$r_i \not\equiv r_j \mod \gcd(a_i, a_j). \tag{7}$$

We now reduce this problem to the integer linear programming problem. We treat $r_1, \ldots, r_k$ as integer variables. In addition to these, for all distinct $i, j \in [k]$, we introduce new integer variables $q_{i,j}$ and $\ell_{i,j}$, and consider the following linear constraints:

$$r_i - r_j = q_{i,j} \cdot \gcd(a_i, a_j) + \ell_{i,j}, \tag{8}$$

$$1 \leq \ell_{i,j} \leq \gcd(a_i, a_j) - 1. \tag{9}$$

Then, we obtain an integer linear program with $O(k^2)$ variables and $O(k^2)$ constraints. Since r_i and r_j satisfy (7) if and only if (8) and (9) hold for some integers $q_{i,j}$ and $\ell_{i,j}$, Exact Pinwheel Packing is reduced to the integer linear program defined by (8) and (9).

It is known that the integer linear programming problem can be solved in FPT time, parameterized by its variable number p; see e.g., [17, 23, 27], and the current smallest parameter dependency is $(\log p)^{O(p)}$ [27]. Using this result, we obtain a $(\log k)^{O(k^2)} \cdot n^{O(1)}$ time algorithm for Exact Pinwheel Packing. ◀

References

1. Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, July 1995. doi:10.1145/210332.210337.
2. Paul Balister. *Erdős Covering Systems*, pages 31–54. London Mathematical Society Lecture Note Series. Cambridge University Press, 2024.
3. Amotz Bar-Noy, Randeep Bhatia, Joseph Naor, and Baruch Schieber. Minimizing service and operation costs of periodic scheduling. *Mathematics of Operations Research*, 27(3):518–544, 2002. doi:10.1287/moor.27.3.518.314.
4. Amotz Bar-Noy and Richard E. Ladner. Windows scheduling problems for broadcast systems. *SIAM Journal on Computing*, 32(4):1091–1113, 2003. doi:10.1137/S009753970240447X.
5. Amotz Bar-Noy, Richard E. Ladner, and Tami Tamir. Windows scheduling as a restricted version of bin packing. *ACM Trans. Algorithms*, 3(3):Article 28, 2007. doi:10.1145/1273340.1273344.
6. Mee Yee Chan and Francis Chin. Schedulers for larger classes of pinwheel instances. *Algorithmica*, 9:425–462, 1993. doi:10.1007/BF01187034.
7. Peter C. Fishburn and J. C. Lagarias. Pinwheel scheduling: Achievable densities. *Algorithmica*, 34(1):14–38, 2002. doi:10.1007/S00453-002-0938-9.
8. M. R. Garey and D. S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness*. W.H. Freeman and Co, 2003.

9 Leszek Gąsieniec, Ralf Klasing, Christos Levcopoulos, Andrzej Lingas, Jie Min, and Tomasz Radzik. Bamboo garden trimming problem (perpetual maintenance of machines with different attendance urgency factors). In Bernhard Steffen, Christel Baier, Mark van den Brand, Johann Eder, Mike Hinchey, and Tiziana Margaria, editors, *SOFSEM 2017: Theory and Practice of Computer Science*, pages 229–240, Cham, 2017. Springer International Publishing. doi:10.1007/978-3-319-51963-0_18.

10 Leszek Gąsieniec, Benjamin Smith, and Sebastian Wild. Towards the 5/6-density conjecture of pinwheel scheduling. In Cynthia A. Phillips and Bettina Speckmann, editors, *Proceedings of the Symposium on Algorithm Engineering and Experiments, ALENEX 2022, Alexandria, VA, USA, January 9-10, 2022*, pages 91–103. SIAM, 2022. doi:10.1137/1.9781611977042.8.

11 Leszek Gąsieniec, Tomasz Jurdziński, Ralf Klasing, Christos Levcopoulos, Andrzej Lingas, Jie Min, and Tomasz Radzik. Perpetual maintenance of machines with different urgency requirements. *Journal of Computer and System Sciences*, 139:103476, 2024. doi:10.1016/j.jcss.2023.103476.

12 Robert Holte, Aloysius Mok, Louis Rosier, Igor Tulchinsky, and Donald Varvel. The pinwheel: a real-time scheduling problem. In *Proceedings of the Twenty-Second Annual Hawaii International Conference on System Sciences. Volume II: Software Track*, volume 2, pages 693–702 vol.2, 1989. doi:10.1109/HICSS.1989.48075.

13 Robert Holte, Louis Rosier, Igor Tulchinsky, and Donald Varvel. Pinwheel scheduling with two distinct numbers. *Theoretical Computer Science*, 100(1):105–135, 1992. doi:10.1016/0304-3975(92)90365-M.

14 Bob Hough. Solution of the minimum modulus problem for covering systems. *Annals of Mathematics*, 181(1):361–382, 2015. doi:10.4007/annals.2015.181.1.6.

15 A.P. Huhn and L. Megyesi. On disjoint residue classes. *Discrete Mathematics*, 41(3):327–330, 1982. doi:10.1016/0012-365X(82)90030-9.

16 Tobias Jacobs and Salvatore Longo. A new perspective on the windows scheduling problem, 2014. arXiv:1410.7237.

17 Ravi Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of Operations Research*, 12(3):415–440, 1987. doi:10.1287/moor.12.3.415.

18 Akitoshi Kawamura. Proof of the density threshold conjecture for pinwheel scheduling. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, STOC 2024, pages 1816–1819, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3618260.3649757.

19 Akitoshi Kawamura. Perpetual scheduling under frequency constraints. In Shin-ichi Minato, Takeaki Uno, Norihito Yasuda, Takashi Horiyama, Ken-ichi Kawarabayashi, Shigeru Yamashita, and Hirotaka Ono, editors, *Algorithmic Foundations for Social Advancement: Recent Progress on Theory and Practice*. Springer, 2025. doi:10.1007/978-981-96-0668-9.

20 Akitoshi Kawamura, Yosuke Kusano, and Yusuke Kobayashi. Pinwheel covering. In *The 14th International Conference on Algorithms and Complexity (CIAC)*, pages 185–199, 2025. doi:10.1007/978-3-031-92935-9_12.

21 Akitoshi Kawamura and Makoto Soejima. Simple strategies versus optimal schedules in multi-agent patrolling. *Theoretical Computer Science*, 839:195–206, 2020. doi:10.1016/j.tcs.2020.07.037.

22 Jan Korst, Emile Aarts, and Jan Karel Lenstra. Scheduling periodic tasks. *INFORMS Journal on Computing*, 8(4):428–435, 1996. doi:10.1287/ijoc.8.4.428.

23 Hendrik W Lenstra Jr. Integer programming with a fixed number of variables. *Mathematics of operations research*, 8(4):538–548, 1983. doi:10.1287/moor.8.4.538.

24 Shun-Shii Lin and Kwei-Jay Lin. A pinwheel scheduler for three distinct numbers with a tight schedulability bound. *Algorithmica*, 19:411–426, 1997. doi:10.1007/PL00009181.

25 Al Mok, Louis Rosier, Igor Tulchinsky, and Donald Varvel. Algorithms and complexity of the periodic maintenance problem. *Microprocessing and Microprogramming*, 27(1):657–664, 1989. Fifteenth EUROMICRO Symposium on Microprocessing and Microprogramming. doi:10.1016/0165-6074(89)90128-2.

26 Moni Naor, Leonard J Schulman, and Aravind Srinivasan. Splitters and near-optimal derandomization. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pages 182–191. IEEE, 1995. doi:10.1109/SFCS.1995.492475.

27 Victor Reis and Thomas Rothvoss. The subspace flatness conjecture and faster integer programming. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 974–988, 2023. doi:10.1109/FOCS57990.2023.00060.

28 Zhi-Wei Sun. On disjoint residue classes. *Discrete Mathematics*, 104(3):321–326, 1992. doi:10.1016/0012-365X(92)90454-N.

29 Martijn van Ee. A 12/7-approximation algorithm for the discrete bamboo garden trimming problem. *Operations Research Letters*, 49(5):645–649, 2021. doi:10.1016/j.orl.2021.07.001.

30 W.D Wei and C.L Liu. On a periodic maintenance problem. *Operations Research Letters*, 2(2):90–93, 1983. doi:10.1016/0167-6377(83)90044-5.

A An FPT Algorithm for Pinwheel Packing

As described in Section 1.3, an FPT algorithm for Pinwheel Packing was already presented in [10] (without an explicit running time bound). In this section, to highlight the similarities and differences between Pinwheel Packing and (Exact) Pinwheel Covering, building on the argument in [10], we present an FPT algorithm for Pinwheel Packing that runs in $O(n) + k^{O(2^k)}$ time.

An instance $(a_i)_{i\in[k]}$ of Pinwheel Packing is called a *strong yes-instance* if there exist pairwise disjoint sets $S_1, \dots, S_k \subseteq \mathbb{Z}$ such that $\mathbb{Z} - \bigcup_{i=1}^{k} S_i$ contains an infinite number of integers. We first present the following claim that was shown in [10] with a different terminology.

▷ Claim 18 (see [10, Proposition 2.1]). Pinwheel Packing can be solved in $\left(\prod_{i=1}^{k} a_i\right)^{O(1)}$ time. We can also check whether a given instance $(a_i)_{i\in[k]}$ is a strong yes-instance in $\left(\prod_{i=1}^{k} a_i\right)^{O(1)}$ time.

Proof of Claim 18. We construct a digraph G with $\prod_{i=1}^{k} a_i$ vertices, each of which corresponds to a k-tuple $(b_1, \dots, b_k)$ with $b_i \in \{0, 1, \dots, a_i - 1\}$ for each $i \in [k]$. This vertex corresponds to a state where task i must be performed within b_i days from the present. We define two types of arcs in G, which we call *normal arcs* and *lazy arcs*. We add a normal arc from a vertex $(b_1, \dots, b_k)$ to a vertex $(b'_1, \dots, b'_k)$ when there is $i \in [k]$ such that $b'_i = a_i - 1$, and $b'_j = b_j - 1$ for $j \in [k] \setminus \{i\}$. We add a lazy arc from a vertex $(b_1, \dots, b_k)$ to a vertex $(b'_1, \dots, b'_k)$ when $b'_j = b_j - 1$ for $j \in [k]$. Note that traversing a normal arc corresponds to performing task i, while traversing a lazy arc corresponds to performing no task. Since a feasible solution of Pinwheel Covering corresponds to an infinite walk in G, A is a yes-instance if and only if G contains a dicycle. Similarly, A is a strong yes-instance if and only if G contains a dicycle containing a lazy arc. Both conditions can be checked in $\left(\prod_{i=1}^{k} a_i\right)^{O(1)}$ time using depth first search. ◁

In a similar way to (Exact) Pinwheel Covering, we prove the following two claims to bound each a_i by a function of k.

▷ Claim 19. Suppose that $A = (a_i)_{i\in[k]}$ is a strong yes-instance of Pinwheel Packing. Then, there exists a feasible solution $(S_i)_{i\in[k]}$ such that at least one integer is not covered by $\bigcup_{i=1}^{k} S_i$ in every consecutive $\prod_{i=1}^{k} a_i$ integers, that is, (4) holds for all $m \in \mathbb{Z}$.

Proof of Claim 19. If A is a strong yes-instance, then the digraph in the proof of Claim 18 has a dicycle containing a lazy arc. Let $(S_i)_{i\in[k]}$ be a feasible solution of Pinwheel Packing that corresponds to an infinite walk traversing such a dicycle repeatedly. Since the length of the dicycle is at most $\prod_{i=1}^{k} a_i$, in every consecutive $\prod_{i=1}^{k} a_i$ steps along this walk, we traverse at least one lazy arc. This means that, in every consecutive $\prod_{i=1}^{k} a_i$ days, there is at least one day on which no task is performed. Therefore, (4) holds for all $m \in \mathbb{Z}$. ◁

▷ Claim 20. Let $A = (a_i)_{i\in[k]}$ be a sequence of positive integers with $a_1 \le a_2 \le \cdots \le a_k$, and let $\ell \in [k-1]$. If $A' = (a_i)_{i\in[\ell]}$ is a strong yes-instance of Pinwheel Packing and $a_{\ell+1} \ge (k-\ell)\prod_{i=1}^{\ell} a_i$, then A is a yes-instance of Pinwheel Packing.

Proof of Claim 20. Suppose that $A' = (a_i)_{i\in[\ell]}$ is a strong yes-instance of Pinwheel Packing and $a_{\ell+1} \ge (k-\ell)\prod_{i=1}^{\ell} a_i$. By Claim 19, there exist pairwise disjoint sets $S_1, \dots, S_\ell \subseteq \mathbb{Z}$ such that at least one integer is not covered by $\bigcup_{i=1}^{\ell} S_i$ in every consecutive $\prod_{i=1}^{\ell} a_i$ integers. Let $(x_j)_{j\in\mathbb{Z}}$ be an infinite sequence of all integers not covered by $\bigcup_{i=1}^{\ell} S_i$, which are arranged in ascending order. For $h \in \{\ell+1, \ell+2, \dots, k\}$, set

$$S_h = \{x_j : j \equiv h \mod k-\ell\}.$$

Then, obviously, $S_1, \dots, S_k$ are pairwise disjoint. For $h \in \{\ell+1, \ell+2, \dots, k\}$, since

$$x_{j+k-\ell} - x_j \le (k-\ell)\prod_{i=1}^{\ell} a_i \le a_{\ell+1} \le a_h$$

holds for every $j \in \mathbb{Z}$, we see that task h is performed every a_h days. Therefore, $(S_i)_{i\in[k]}$ is a feasible solution for the instance A. ◁

We are now ready to show the following theorem.

▶ **Theorem 21** (see [10, Corollary 3.2]). *Pinwheel Packing can be solved in $O(n) + k^{O(2^k)}$ time, where $n := \sum_{i=1}^{k} \log a_i$ is the input size of the problem.*

Proof. Let $A = (a_i)_{i\in[k]}$ be an instance of Pinwheel Packing. Without loss of generality, we may assume that $a_1 \le a_2 \le \cdots \le a_k$. If $a_1 \ge k$, then A is a yes-instance, as we can simply perform each task exactly once every k days. Thus, it suffices to consider the case where $a_1 < k$.

If $a_i \le k^{2^{i-1}}$ for every $i \in [k]$, then we can check the feasibility of A in $\left(\prod_{i=1}^{k} a_i\right)^{O(1)} = k^{O(2^k)}$ time by Claim 18. Otherwise, let $\ell \in [k-1]$ be the minimum integer with $a_{\ell+1} > k^{2^\ell}$, which can be computed in $O(n)$ time. Then, we obtain

$$a_{\ell+1} > k^{2^\ell} = k\prod_{i=1}^{\ell} k^{2^{i-1}} > (k-\ell)\prod_{i=1}^{\ell} a_i \tag{10}$$

in the same way as (5). We observe that $(a_i)_{i\in[k]}$ is a yes-instance if and only if $(a_i)_{i\in[\ell]}$ is a strong yes-instance, as the "only if" part is trivial and the "if" part follows from Claim 20 and (10). Therefore, it suffices to determine whether $(a_i)_{i\in[\ell]}$ is a strong yes-instance or not, which can be done in $\left(\prod_{i=1}^{\ell} a_i\right)^{O(1)}$ time by Claim 18. This running time is bounded by $k^{O(2^k)}$, because $a_i \le k^{2^{i-1}}$ for $i \in [\ell]$. ◀

Quadratic Kernel for Cliques or Trees Vertex Deletion

Soh Kumabe
CyberAgent, Tokyo, Japan

Abstract

We consider Cliques or Trees Vertex Deletion, which is a hybrid of two fundamental parameterized problems: Cluster Vertex Deletion and Feedback Vertex Set. In this problem, we are given an undirected graph G and an integer k, and asked to find a vertex subset X of size at most k such that each connected component of $G - X$ is either a clique or a tree. Jacob et al. (ISAAC, 2024) provided a kernel of $O(k^5)$ vertices for this problem, which was recently improved to $O(k^4)$ by Tsur (IPL, 2025).

Our main result is a kernel of $O(k^2)$ vertices. This result closes the gap between the kernelization result for Feedback Vertex Set, which corresponds to the case where each connected component of $G - X$ must be a tree.

Although both *cluster vertex deletion number* and *feedback vertex set number* are well-studied structural parameters, little attention has been given to parameters that generalize both of them. In fact, the lowest common well-known generalization of them is clique-width, which is a highly general parameter. To fill the gap here, we initiate the study of the *cliques or trees vertex deletion number* as a structural parameter. We prove that Longest Cycle, which is a fundamental problem that does not admit $o(n^k)$-time algorithm unless ETH fails when k is the clique-width, becomes fixed-parameter tractable when parameterized by the cliques or trees vertex deletion number.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases Fixed-Parameter Tractability, Kernelization, Deletion to Scattered Graph Classes, Cluster Vertex Deletion, Feedback Vertex Set

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.48

Related Version *Full Version*: https://arxiv.org/abs/2509.16815

1 Introduction

Given a graph, can we remove at most k vertices so that the resulting graph belongs to a class Π of well-structured graphs? Such problems are called *vertex deletion problems* and include many well-studied parameterized problems. Indeed, Vertex Cover [12, 23], Feedback Vertex Set [11, 25, 34, 39], Cluster Vertex Deletion [4, 15, 24], Odd Cycle Transversal [32, 38], Interval Vertex Deletion [6], and Chordal Vertex Deletion [36] correspond to the cases where Π is the classes of collection of isolated vertices, collection of trees, collection of cliques, bipartite graphs, interval graphs, and chordal graphs, respectively.

However, why must Π be a single graph class? It is still reasonable to call a graph well-structured when all connected components are well-structured, even if different components belong to different classes. Jacob et al. [26] introduced the problem framework of *deletion to scattered graph classes* capturing this concept, asking: Can we remove at most k vertices from a given graph so that each connected component of the resulting graph belongs to one of the graph classes $(\Pi_1, \ldots, \Pi_p)$? Together with the subsequent paper [27], they investigated the parameterized complexity of this type of problems and provided both general and problem-specific algorithms.

This paper considers the following fundamental special case of deletion problems to scattered graph classes, called Cliques or Trees Vertex Deletion, which is first studied in [27].

© Soh Kumabe;
licensed under Creative Commons License CC-BY 4.0

> Cliques or Trees Vertex Deletion: Given an undirected graph $G = (V, E)$ and an integer $k \in \mathbb{Z}_{\geq 1}$, is there a vertex subset $X \subseteq V$ with $|X| \leq k$ such that each connected component of $G - X$ is either a clique or a tree?

This case is particularly interesting because it combines two of the most prominent parameterized problems, Feedback Vertex Set and Cluster Vertex Deletion. Moreover, since both the *feedback vertex set number* and the *cluster vertex deletion number* are well-studied structural parameters, it is natural to expect that the *cliques or trees vertex deletion number*, which is the minimum k such that (G, k) becomes an yes-instance of Cliques or Trees Vertex Deletion, is likewise an interesting structural parameter. In particular, this number captures the structural simplicity of graphs that contain both dense parts (i.e., clique components) and sparse parts (i.e., tree components). This illustrates the benefit of considering deletion to scattered graph classes, as a deletion distance to a single dense or sparse class alone cannot capture such a property.

It has been proved that this problem is in FPT [26, 27]; that is, it admits an $f(k)n^c$-time algorithm for some computable function f and constant c. Accordingly, researchers are investigating kernelization algorithms, which are polynomial-time algorithms that reduce the given instance to an equivalent instance of size $g(k)$ for some function g. Jacob et al. [28] showed that this problem admits a kernel with $O(k^5)$ vertices, which was later improved to $O(k^4)$ vertices by Tsur [40]. However, there is still a significant gap when compared to the kernelization results of the original two problems, Feedback Vertex Set and Cluster Vertex Deletion, which admit a kernel with $O(k^2)$ vertices [25, 39] and $O(k)$ vertices [4], respectively.

In this paper, we close this gap by proving the following.

▶ **Theorem 1.** *Cliques or Trees Vertex Deletion admits a kernel with $O(k^2)$ vertices.*

We remark that our result may surpass the expectations in the original research by Jacob et al. [28], as they stated the following in their conclusion section:

> One natural open question is to improve the size of our kernel, e.g. to $O(k^3)$ vertices. We believe that such a result is possible to achieve, but we suspect that it would require new techniques to develop such results.

We further initiate the study of Cliques or Trees Vertex Deletion as a structural parameter. As mentioned above, both the feedback vertex set number [18, 41] and the cluster vertex deletion number [10, 20] are well-studied structural parameters. However, their common generalization has received relatively little attention. Actually, the smallest well-investigated class that includes both graphs with bounded feedback vertex set number and graphs with bounded cluster vertex deletion number is the class of bounded *clique-width*, which is a very general parameter often placed at the top of diagrams illustrating the inclusion relationships among structural parameters of graphs (see Figure 1).

As the second contribution of this paper, we demonstrate that the cliques or trees vertex deletion number is indeed a useful structural parameter. Particularly, we demonstrate that when parameterized by the cliques or trees vertex deletion number, a fundamental problem that is hard when parameterized by clique-width becomes tractable. Specifically, we consider the following Longest Cycle, one of the most well-investigated problems in the field of parameterized complexity under structural parameterizations [3, 8, 9, 10, 13, 14, 19, 21, 29, 33].

> Longest Cycle: Given an undirected graph $G = (V, E)$, find a cycle of G with the largest possible number of vertices.

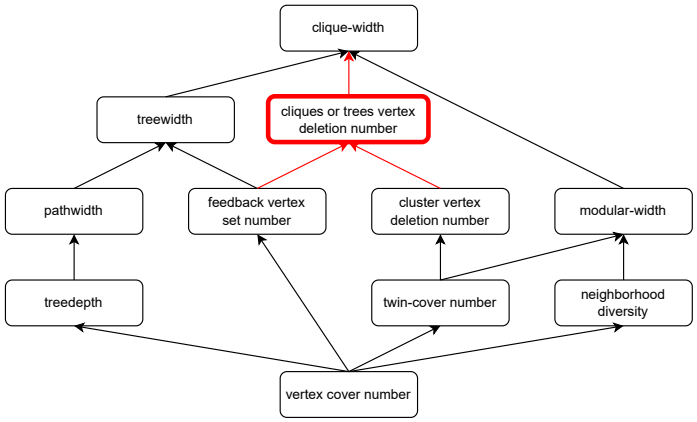


Figure 1 Diagram of structural parameters including the cliques or trees vertex deletion number.

It is known that, when k is the clique-width, Longest Cycle (and even the special case Hamiltonian Cycle) does not admit an $n^{o(k)}$-time algorithm unless ETH fails [13, 14]. We prove the following.

▶ **Theorem 2.** *Longest Cycle can be solved in $2^{O(k \log k)}$-time, where k is the cliques or trees vertex deletion number.*

Note that this result generalizes, without losing time complexity, the FPT algorithm for Hamiltonian Cycle parameterized by the cluster vertex deletion number given by Doucha and Kratochvíl [10] in the following two aspects. First, we solve Longest Cycle, which is a generalization of Hamiltonian Cycle. Second, our algorithm is parameterized by cliques or trees vertex deletion number, which is a more general parameter. We further note that it is not hard to see that the clique-width of graphs with bounded cliques or trees vertex deletion number is bounded.

1.1 Related Work

The literature on Π-deletion problems has studied several variants of Feedback Vertex Set and Cluster Vertex Deletion. Examples of extensions of Feedback Vertex Set are the cases where Π consists of graphs with treewidth at most η [16, 31] and graphs that are at most l edges away from forests [37]. Examples of variants of Cluster Vertex Deletion are the cases where Π consists of s-plexes [35], s-clubs [22], and graphs with a small dominating set number [2].

Jacob et al. [26] introduced the notion of *deletion to scattered graph classes*. In that paper, they obtained two general tractability results, stating that $(\Pi_1, \ldots, \Pi_p)$-Vertex Deletion (i) is fixed-parameter tractable when each Π_i-Vertex Deletion is fixed-parameter tractable, and (ii) admits a $2^{poly(k)}$-time algorithm when each Π_i is defined by a finite set of forbidden subgraphs. Jansen et al. [30] improved the time complexity of result (ii) to $2^{O(k)}$. In the subsequent work [27], Jacob et al. considered several specific cases and obtained efficient FPT algorithms and approximation algorithms. In particular, for Cliques or Trees Vertex Deletion, they obtained an $O^*(4^k)$-time FPT algorithm and a polynomial-time 4-approximation algorithm. Jacob et al. [28] provided a kernel with $O(k^5)$ vertices for this problem. Tsur [40] improved both of FPT and kernelization results by presenting a deterministic algorithm running in $O(3.46^k)$ time, a randomized algorithm running in $O(3.103^k)$ time, and a kernel with $O(k^4)$ vertices.

Several studies have investigated the parameterized complexity of Longest Cycle and its special case, Hamiltonian Cycle, under structural parameterizations. Parameterized by the cluster vertex deletion number, Doucha and Kratochvíl [10] presented an FPT algorithm for Hamiltonian Cycle. Parameterized by the clique-width, Fomin et al. [13, 14] showed that no $n^{o(k)}$-time algorithm for Hamiltonian Cycle exists unless ETH fails. Bergougnoux et al. [3] proved this bound is tight by giving an $n^{O(k)}$-time algorithm. The cases parameterized by treewidth [9], pathwidth [8], twin-cover number [19], and proper interval deletion number [21] are also investigated, as well as directed tree-width [33] and directed feedback vertex set number [29] for the directed version.

1.2 Technical Overview

Here, we provide a technical overview of Theorems 1 and 2. Section 1.2.1 also contains a brief explanation of how our kernelization algorithm differs from the existing kernelization algorithms by Jacob et al. [28] and Tsur [40].

1.2.1 Overview of Theorem 1

Here, we briefly explain our idea toward Theorem 1. Our approach, at the highest level, is based on the following simple observation. Let v be a vertex, and let $N_G(v) := \{u \in V : (u, v) \in E\}$ be the set of neighbors of v. Assume there is a feasible solution[1] X such that v is in a connected component of $G-X$ that is a clique. Then, the neighbors of v in $G-X$ induce a clique. In particular, $N_G(v)$ contains a clique of size at least $|N_G(v) \setminus X| \geq |N_G(v)| - k$. Similarly, assume there is a feasible solution X such that v is in a connected component of $G - X$ that is a tree. Then, the neighbors of v in $G - X$ induce an independent set. In particular, $N_G(v)$ contains an independent set of size at least $|N_G(v) \setminus X| \geq |N_G(v)| - k$.

The core observation is that if $|N_G(v)|$ is large, say at least $2k + 2$, these two situations cannot occur simultaneously, as a clique and an independent set cannot share an edge. In other words, if the degree of v is large, we can determine that v is either

- always in a clique component of $G - X$ unless $v \in X$, or
- always in a tree component of $G - X$ unless $v \in X$,

for all feasible solutions X.

In our algorithm, we first partition the vertices into three subsets. Let $c = 7$. Set V_{ld} of vertices with a high degree ($> ck$) and dense neighbors, set V_{ls} of vertices with a high degree ($> ck$) and sparse neighbors, and set V_{small} of vertices with a low degree ($\leq ck$). Using the above observation, we can claim that, for any feasible solution X, vertices in V_{ld} cannot be in a tree component of $G - X$, and vertices in V_{ls} cannot be in a clique component of G of $G - X$. This observation, very roughly speaking, enables us to apply reduction rules to V_{ld} as if we were solving Cluster Vertex Deletion. Similarly, we can apply reduction rules to V_{ls} as if we were solving Feedback Vertex Set. Furthermore, we can reduce the number of vertices in V_{small} using the fact that they have small degrees.

Here we go into a little more detail. Our kernelization algorithm proceeds as follows. First, we reduce the degrees of vertices in V_{ls} into $O(k)$. To acheive this, we directly apply reduction rules used in the celebrated quadratic kernelization of Feedback Vertex Set by Thomassé [39]. Note that this direct applicability is already a benefit of partitioning the vertex set. Indeed, the original kernelization algorithm by Jacob et al. [28] also relied

[1] We say a solution X is *feasible* if $|X| \leq k$ and each connected component of $G - X$ is either a clique or a tree.

on Thomassé's quadratic kernelization in its "tree part" (and Tsur's improvement [40] did not touch this part), but extended the algorithm and analysis, including the use of a new version of the expansion lemma by Fomin et al. [15]. In contrast, ours is almost identical to Thomassé's, and its analysis is also nearly the same except for a few technical details, making our approach significantly simpler.

Next, we reduce the number of vertices in V_{ld}. The main technical contribution of this paper lies in this part, since the kernelization algorithms by Jacob et al. [28] and Tsur [40] bound the number of vertices in the "clique part" by $O(k^5)$ and $O(k^4)$, respectively, which dominates the overall kernel size. Both of those algorithms are based on a marking procedure, whereas ours is not. As in those algorithms, we first compute a constant factor approximate solution S in polynomial time and reduce the number of clique components of $G - S$ by $O(k)$ using the textbook reduction [17] for Cluster Vertex Deletion using the expansion lemma. Now, we can argue that there are only $O(k^2)$ vertices in V_{ld} that are adjacent to some vertex outside V_{ld} as follows. First, any such vertex must

- **(i)** belongs to S,
- **(ii)** be adjacent to a vertex in $S \setminus V_{\mathrm{ld}}$, or
- **(iii)** belongs to a clique component of $G - S$ that contains a vertex outside V_{ld}.

The number of vertices satisfying (i) is $O(k)$ by definition. Since every vertex outside V_{ld} has degree $O(k)$, the number of vertices satisfying (ii) is bounded by $O(k^2)$. Furthermore, each clique component that appears in (iii) contains at most $O(k)$ vertices because it contains a vertex with degree $O(k)$. Together with the fact that the number of clique components is $O(k)$, the number of vertices satisfying (iii) is bounded by $O(k^2)$. Now, we concentrate on reducing the number of vertices in V_{ld} that are adjacent only to vertices in V_{ld} by $O(k^2)$. To do this, we first borrow ideas from the quadratic kernelization of the 3-Hitting Set by Abu-Khzam [1] and construct a list $\mathcal{P}$ of induced P_3s (that are, induced subgraphs that are paths of length two) such that no two different P_3 share more than one vertex. A standard argument similar to that used in Buss Kernel [5] for Vertex Cover reduces the size of $\mathcal{P}$ to $O(k^2)$. Let $V_{\mathrm{ldmod}} := V_{\mathrm{ld}} \setminus \bigcup_{P \in \mathcal{P}} P$. By introducing additional structural observations, we can state that

- there are at most $O(k)$ connected components in V_{ldmod}, and
- each connected component C of V_{ldmod} is a *clique-module* of V_{ld}, that is, the set $N_{V_{\mathrm{ld}}}(v) \cup \{v\}$ is same for all $v \in C$ and includes C.

The important observation is that if G contains a clique-module of V of size at least $k + 4$, then one of its vertices can be safely removed. This rule seems to bound the size of each clique-module by $O(k)$ and bound the size of V_{ldmod} by $O(k^2)$. However, it is still insufficient because we want clique-modules of V for the reduction, whereas the observation above only provides clique-modules of V_{ld}. Nevertheless, if a clique-module in V_{ld} consists only of vertices with no neighbor outside V_{ld}, then it is also a clique-module of V. Therefore, we can reduce the number of vertices in each clique-module that have no neighbor outside V_{ld} to $O(k)$, obtaining the desired bound. (Precisely speaking, we need to perform a slightly more careful argument to deal with multi-edges that may be introduced in reductions for V_{ls}.)

We reduce the number of vertices in $V \setminus V_{\mathrm{ld}}$ to $O(k^2)$ to complete the kernelization. By an argument similar to that for (iii) above, we can bound the number of vertices in $V \setminus V_{\mathrm{ld}}$ that belong to clique components of $G - S$ by $O(k^2)$. Therefore, it remains to bound the number of vertices that belong to tree components. To do this, we apply a few reduction rules based on local structure. Most of these rules appear in [28], but one is new. We apply the following standard argument used in the kernelization of Feedback Vertex Set: If a graph with minimum degree at least 3 can be turned into a forest by removing k vertices of

degree at most t, then the number of vertices in the original graph was $O(tk)$. However, in this case, we cannot completely eliminate vertices of degree 1 or 2, so this argument cannot be applied directly. Nevertheless, we can bound the number of such low-degree vertices using the numbers of vertices of degree 3 and at least 4. By extending the argument for Feedback Vertex Set using these fine-grained bounds, we can bound the number of vertices in $V \setminus V_{\mathrm{ld}}$ by $O(k^2)$, which completes the analysis.

1.3 Overview of Theorem 2

Here, we briefly explain our ideas toward Theorem 2. Let $G = (V, E)$ be a graph and $X \subseteq V$ be a given vertex subset with $|X| = k$ such that each connected component of $G - X$ is either a clique or a tree. We begin by brute-force the order in which the desired cycle P visits the vertices in X. If P is disjoint from X, the problem is trivial, so we may assume that P intersects X, and denote the vertices in $X \cap P$ by $v_1, \ldots, v_l$ in the order they appear along P. Let P_i be the v_i, v_{i+1}-path appearing on P (indices modulo l). Then, unless P_i has length 1, the internal vertices of P_i belong to a single connected component C_i of $G - X$.

The next step is limiting the candidates of C_i. Here, we can state that, as candidates of C_i, it is sufficient to consider the top k components that admit the longest v_i, v_{i+1}-paths. Then, we can brute-force over all tuples $(C_1, \ldots, C_l)$ within a total cost of $2^{O(k \log k)}$. Now, the problem is reduced to solving the following problem for each connected component H of $G - X$, where we set $J_H := \{i \colon C_i = H\}$.

> Given a list of vertex subset pairs $\{(V_{i1} := N(v_i) \cap C_i,\ V_{i2} := N(v_{i+1}) \cap C_i)\}_{i \in J_H}$, compute the maximum total length of vertex-disjoint paths $\{P_i\}_{i \in J_H}$ such that each P_i is a path from a vertex in V_{i1} to a vertex in V_{i2}.

We give FPT algorithms for this problem on both cliques and trees. First, we explain the algorithm for cliques. We brute-force over flags $f \in \{0, 1\}^{J_H}$, where $f_i = 0$ represents that P_i consists of a single vertex, and $f_i = 1$ indicates that P_i contains at least two vertices. The problem of whether a family of paths satisfying the conditions defined by f exists can be reduced to the bipartite matching problem and, thus, solved in polynomial time. If such a family of paths exists for some flag f such that $f_i = 1$ holds for some $i \in J_H$, we can extend the family to cover all vertices in H by appropriately adding internal vertices to the paths. Thus, in this case, the answer is $|H| - |J_H|$. If such a family of paths exists only for $f = (0, \ldots, 0)$, the answer is zero. If no such family of paths exists for any f, then the problem is infeasible. The time complexity is $O^*(2^k)$.

Now, we explain the algorithm for trees. Our algorithm uses dynamic programming. We omit the formal details here, but the intuition is as follows. Regard H as a rooted tree. For a vertex v and a set $Z \subseteq J_H$, we define $\mathsf{DP}[v][Z]$ denote the maximum total length of a family of paths $\{P_i\}_{i \in Z}$ that can be packed into the subtree rooted at v. We compute these values in a bottom-up manner. We can analyze that such dynamic programming can be implemented to work in $O^*(3^k)$ time.

1.4 Organization

The rest of this paper is organized as follows. In Section 2, we introduce basic notation and well-known techniques in the literature on kernelization. In Section 3, we prove Theorem 1 by constructing a kernel for Cliques or Trees Vertex Deletion with $O(k^2)$ vertices. Due to the space limitation, some proofs are given in the full version. In the full version, we prove Theorem 2 by presenting an FPT algorithm for Longest Cycle parameterized by the cliques or trees vertex deletion number.

2 Preliminaries

In this paper, the term *graph* refers to an undirected graph, which does not contain self-loops but may contain multi-edges. For a graph $G = (V, E)$ and a vertex $v \in V$, we call a vertex belonging to $N_G(v) := \{u \in V \colon (u, v) \in E\}$ a *neighbor* of v in G, and the number of edges incident to v, denoted $d_G(v)$, is referred to as the *degree* of v. Note that $|N_G(v)|$ and $d_G(v)$ may differ due to multi-edges. Moreover, we denote by $\rho_G(v)$ the size of the set $\{\{u_1, u_2\} \subseteq N_G(v) \colon (u_1, u_2) \in E\}$. In other words, $\rho_G(v)$ represents the number of edges connecting the neighbors of v in G, where multi-edges are counted as a single edge. When the context is clear, we omit the subscript G and write $N(v)$, $d(v)$, and $\rho(v)$ for simplicity. A vertex v with $d(v) = 1$ is called a *pendant*.

For a vertex subset $Z \subseteq V$, we denote $E(Z) := \{e \in E \colon e \subseteq Z\}$. The *subgraph of G induced by Z* is the graph $(Z, E(Z))$. For simplicity, when there is no risk of confusion, we identify the vertex set Z with the subgraph induced by it: that is, when we refer to the "graph Z" for $Z \subseteq V$, we mean the subgraph induced by Z. Moreover, for a vertex subset $Z \subseteq V$, we write $G - Z$ to denote the subgraph induced by $V \setminus Z$. When Z consists of a single vertex z, we abbreviate $G - \{z\}$ as $G - z$. Similarly, we denote by $G - e$ the graph obtained by removing an edge $e \in E$ from G. A vertex subset $Z \subseteq V$ induces a *clique* if there is exactly one edge between any two different vertices in Z, an *independent set* if there is no edge between any two vertices in Z, and a *tree* if Z is connected and contains no cycle.

For a vertex $v \in V$ and $t \in \mathbb{Z}_{\geq 1}$, a *$v$-flower* of *order* t is a set of t cycles passing through v such that no two cycles share a common vertex other than v. The following is well-known.

▶ **Lemma 3** (Gallai's Theorem [7, 17, 39]). *Given an undirected graph $G = (V, E)$, a vertex $v \in V$, and an integer $t \in \mathbb{Z}_{\geq 1}$, there is a polynomial-time algorithm that computes either*

- *a v-flower of order $t + 1$, or*
- *a vertex set $B \subseteq V \setminus \{v\}$ of size at most $2t$ such that $G - B$ contains no cycle passing through v.*

For vertex subsets K, L and an integer $q \in \mathbb{Z}_{\geq 1}$, an edge set M is a *q-expansion of K into L* if

- exactly q edges of M are incident to each vertex in K, and
- exactly one edge of M is incident to each vertex in L.

The following lemma is also well-known in the literature on kernelization.

▶ **Lemma 4** (Expansion Lemma [7, 17]). *Let $H := (K \dot\cup L, E)$ be a bipartite graph with vertex bipartition (K, L) and $q \in \mathbb{Z}_{\geq 1}$. Assume $|L| \geq q|K|$ and L contains no isolated vertex. Then, there is a polynomial-time algorithm that computes a pair of non-empty vertex subsets $K' \subseteq K$ and $L' \subseteq L$ such that*

- *$N_H(L') \subseteq K'$, and*
- *there exists a q-expansion of K' into L'.*

3 Quadratic Kernel for Cliques or Trees Vertex Deletion

Let $G = (V, E)$ be a graph and $k \in \mathbb{Z}_{\geq 1}$. In this section, we prove Theorem 1 by constructing a quadratic kernel for Cliques or Trees Vertex Deletion. Due to the space limitation, we defer the proofs of the lemmas marked with an asterisk to the full version.

3.1 Partitioning Vertices

We begin by classifying the vertices into three categories. Let

$$V_{\mathrm{ls}} := \left\{ v \in V : |N(v)| > 7k \wedge \rho(v) \leq \frac{|N(v)|(|N(v)|-1)}{4} \right\},$$
$$V_{\mathrm{ld}} := \left\{ v \in V : |N(v)| > 7k \wedge \rho(v) > \frac{|N(v)|(|N(v)|-1)}{4} \right\},$$
$$V_{\mathrm{small}} := \{ v \in V : |N(v)| \leq 7k \}.$$

The vertices belonging to the first, second, and third groups are referred to as *large-sparse*, *large-dense*, and *small*, respectively. The following lemma states that, for any feasible solution X, large-sparse vertices are included in either X or a tree component of $G - X$.

▶ **Lemma 5 (*).** *Let $v \in V_{\mathrm{ls}}$. Then, for any feasible solution X with $v \notin X$, v is in a tree component of $G - X$.*

The following lemma states a result symmetric to Lemma 5, that is, for any feasible solution X, large-dense vertices are included in either X or a clique component of $G - X$.

▶ **Lemma 6 (*).** *Let $v \in V_{\mathrm{ld}}$. Then, for any feasible solution X with $v \notin X$, v is in a clique component of $G - X$.*

Several times throughout this paper, we use the following type of alternative evidence for a vertex belonging to a clique component or a tree component.

▶ **Lemma 7 (*).** *Let $v \in V$ and X be a feasible solution with $v \notin X$. If $N(v)$ contains a clique of size $k+2$, then v is in a clique component of $G - X$. Similarly, if $N(v)$ contains an independent set of size $k+2$, then v is in a tree component of $G - X$.*

In the rest of this paper, we will use Lemmas 5, 6, and 7 as basic tools without specifically mentioning them.

3.2 Bounding Sizes of Neighbors of Vertices in V_{ls}

In this section, we reduce the size of $N(v)$ for large-sparse vertices $v \in V_{\mathrm{ls}}$. Most of the reduction rules in this section are the same as the quadratic kernelization of Feedback Vertex Set by Thomassé [39], while some details in the analysis require additional care in proofs. We begin with the following.

▶ **Reduction Rule 1.** *Let v be any large-sparse vertex. Apply Lemma 3 for v and $t = k$. If a v-flower of order $k+1$ is found, remove v and decrease k by 1.*

▶ **Lemma 8 (*).** *Reduction Rule 1 is safe.*

Let v be a large-sparse vertex and assume Lemma 3 finds a vertex set B with size at most $2k$ that hits all cycles containing v. Let $\mathcal{C}_{\mathrm{tree}}$ be the family of connected components of $G - v - B$ that are trees and adjacent to v. Similarly, let $\mathcal{C}_{\mathrm{nontree}}$ be the family of connected components of $G - v - B$ that are not trees and adjacent to v. Since $G - B$ contains no cycle containing v, for each $C \in \mathcal{C}_{\mathrm{tree}} \cup \mathcal{C}_{\mathrm{nontree}}$, we have $|N(v) \cap C| = 1$. Particularly, $|N(v)| = |\mathcal{C}_{\mathrm{tree}}| + |\mathcal{C}_{\mathrm{nontree}}| + |N(v) \cap B|$. We bound $|N(v)|$ by bounding these three terms. Obviously, $|N(v) \cap B| \leq |B| \leq 2k$. To bound $|\mathcal{C}_{\mathrm{nontree}}|$, we use the following reduction rule.

▶ **Reduction Rule 2.** *If $|\mathcal{C}_{\mathrm{nontree}}| \geq k+1$, remove v and decrease k by 1.*

▶ **Lemma 9** (*). *Reduction Rule 2 is safe.*

Now we bound $|\mathcal{C}_{\mathrm{tree}}|$ by $4k$ using the expansion lemma. We construct an auxiliary bipartite graph H. The vertex set of H is $B \dot\cup \mathcal{C}_{\mathrm{tree}}$ with bipartition $(B, \mathcal{C}_{\mathrm{tree}})$. We add an edge between $b \in B$ and $C \in \mathcal{C}_{\mathrm{tree}}$ if and only if $N_G(b) \cap C \neq \emptyset$. We use the following reduction rule to ensure the part $\mathcal{C}_{\mathrm{tree}}$ does not contain isolated vertices.

▶ **Reduction Rule 3.** *If there is a component $C \in \mathcal{C}_{\mathrm{tree}}$ that has no neighbor in B, remove all vertices of C.*

▶ **Lemma 10** (*). *Reduction Rule 3 is safe.*

Assume $|\mathcal{C}_{\mathrm{tree}}| \geq 4k$. We apply 2-expansion lemma to H and obtain vertex sets $\mathcal{C}' \subseteq \mathcal{C}_{\mathrm{tree}}$ and $B' \subseteq B$ such that there is a 2-expansion of B' into $\mathcal{C}'$. We have $|B'| \leq k$ because otherwise we obtain a v-flower of order $k+1$. We apply the following reduction.

▶ **Reduction Rule 4.** *Remove each edge between v and $\mathcal{C}'$. Then, connect v and each vertex in B' by a double-edge.*

▶ **Lemma 11** (*). *Assume $|\mathcal{C}_{\mathrm{tree}}| \geq 4k$. Then, Reduction Rule 4 is safe.*

Since all the above reduction rules reduce the number of pairs of vertices connected by at least one edge, the reduction rules in this section can be applied only a polynomial number of times. Thus, we have the following.

▶ **Lemma 12** (*). *Let G be the graph obtained by exhaustively applying all the above reduction rules. Then, for all vertex $v \in V \setminus V_{\mathrm{ld}}$, we have $|N(v)| \leq 7k$ (and thus, $V_{\mathrm{ls}} = \emptyset$).*

3.3 Bounding $|V_{\mathrm{ld}}|$

In this section, we reduce the number of vertices in V_{ld}. This part is the main technical contribution of this paper. As in [28], we apply a 4-approximation algorithm for Cliques or Trees Vertex Deletion given in [27] and obtain an approximate solution $S \subseteq V$. We apply the following reduction rule to ensure $|S| \leq 4k$, which is clearly safe.

▶ **Reduction Rule 5.** *If $|S| > 4k$, return* no.

Let $\mathcal{C}_{\mathrm{clique}}$ be the family of connected components of $G - S$ that are cliques of size at least 3. Similarly, let $\mathcal{C}_{\mathrm{tree}}$ be the family of connected components of $G - S$ that are trees. The following rule ensures that each vertex $v \in V_{\mathrm{ld}}$ is contained in a clique component of $G - S$ unless $v \in S$.

▶ **Reduction Rule 6.** *If there is a vertex $v \in V_{\mathrm{ld}}$ that is in a tree component of $G - S$, remove v and decrease k by 1.*

▶ **Lemma 13.** *Reduction Rule 6 is safe.*

Proof. We prove that v is contained in all solutions X. Since $v \in V_{\mathrm{ld}}$, v is in a clique component of $G - X$ unless $v \in X$. Since v is in a tree component of $G - S$, $N_{G-S}(v)$ is an independent set, whose size is at least $|N_G(v)| - |S| \geq 7k - 4k = 3k \geq k + 2$. Thus, v is in a tree component of $G - X$ unless $v \in X$, which leads to $v \in X$. ◀

We construct an auxiliary bipartite graph H as follows. The vertex set of H is $S \dot\cup \mathcal{C}_{\mathrm{clique}}$ with bipartition $(S, \mathcal{C}_{\mathrm{clique}})$. We add an edge between $s \in S$ and $C \in \mathcal{C}_{\mathrm{clique}}$ if and only if $N_G(s) \cap C \neq \emptyset$. To ensure the part $\mathcal{C}_{\mathrm{clique}}$ does not contain isolated vertices, we apply the following fundamental rule, which is clearly safe.

▶ **Reduction Rule 7.** *If G contains a connected component that is a clique or a tree, remove that component.*

Assume $|\mathcal{C}_{\text{clique}}| \geq 2|S|$. We apply 2-expansion lemma to H and obtain vertex sets $\mathcal{C}' \subseteq \mathcal{C}_{\text{clique}}$ and $S' \subseteq S$. The following reduction is proved to be safe in [28].

▶ **Reduction Rule 8.** *Remove vertices of S' and decrease k by $|S'|$.*

▶ **Lemma 14** ([28])**.** *Reduction Rule 8 is safe.*

Now we can assume $|\mathcal{C}_{\text{clique}}| \leq 8k$. We first bound the number of vertices in V_{ld} that are adjacent to some vertices outside V_{ld}. We have the following.

▶ **Lemma 15.** *After applying all the above reduction rules, there are at most $84k^2 + 4k$ vertices in V_{ld} that have some neighbor outside V_{ld}.*

Proof. A large-dense vertex v is adjacent to a vertex outside V_{ld} only when
(i) $v \in S$,
(ii) $v \notin S$ and v has a neighbor in $S \setminus V_{\text{ld}}$, or
(iii) $v \notin S$ and the component $C \in \mathcal{C}_{\text{clique}}$ containing v contains a vertex from $V \setminus V_{\text{ld}}$.
Reduction Rule 5 ensures that at most $|S| \leq 4k$ vertices satisfy condition (i). Moreover, since vertices in $V \setminus V_{\text{ld}}$ has degree at most $7k$, at most $7k|S| \leq 28k^2$ vertices satisfy condition (ii). Furthermore, if v satisfies condition (iii), C is a clique of size at most $7k + 1$ because it contains a vertex with degree at most $7k$. Therefore, at most $7k|\mathcal{C}_{\text{clique}}| \leq 56k^2$ vertices satisfy condition (iii). ◀

Now we bound the number of vertices $v \in V_{\text{ld}}$ with $N_G(v) \subseteq V_{\text{ld}}$. A vertex triplet (v_1, v_2, v_3) *induces* P_3 if $(v_1, v_2), (v_2, v_3) \in E$ and $(v_3, v_1) \notin E$. We remark that even if the edges (v_1, v_2) or (v_2, v_3) are multi-edges, we still consider them as inducing a P_3. The following lemma is analogous to the fundamental observation in the literature on Cluster Vertex Deletion.

▶ **Lemma 16.** *Let X be a feasible solution and P be a subset of V_{ld} that induces P_3. Then, $X \cap P \neq \emptyset$.*

Proof. Assume otherwise and let C be the connected component of $G - X$ that contains P. From Lemma 6, C is a clique component. However, cliques cannot contain induced P_3, leading to a contradiction. ◀

This leads to the following reduction rule, which is clearly safe.

▶ **Reduction Rule 9.** *Let $v \in V_{\text{ld}}$. If there is a collection of $k + 1$ induced P_3s in V_{ld} such that any two of them intersect only at $\{v\}$, then remove v and decrease k by 1.*

Whether $v \in V$ satisfies the condition of Reduction Rule 9 can be checked by computing the maximum matching on the graph $(N_G(v), E_v)$, where E_v is the set of pairs of the vertices (u_1, u_2) such that $\{v, u_1, u_2\}$ induces P_3. Therefore, this rule can be applied in polynomial time. Let $\mathcal{P}$ be a maximal collection of induced P_3s in V_{ld} such that any two induced P_3s in the collection have an intersection of size at most 1. The idea to construct this $\mathcal{P}$ and the following reduction rule are borrowed from the quadratic kernelization of 3-Hitting Set by Abu-Khzam [1].

▶ **Reduction Rule 10.** *If $|\mathcal{P}| > k^2$, return* no.

▶ **Lemma 17.** *Reduction Rule 10 is safe.*

Proof. Assume $|\mathcal{P}| > k^2$ and let X be a feasible solution. From the assumption that Reduction Rule 9 cannot be applied, each vertex in X hits at most k induced P_3s in $\mathcal{P}$. Therefore, there exists an induced P_3 that is disjoint from X, which contradicts Lemma 16. ◀

Now we can assume $\left|\bigcup_{P\in\mathcal{P}} P\right| \leq 3|\mathcal{P}| \leq 3k^2$. The remaining task is to bound the number of vertices in $V_{\mathrm{ldmod}} := V_{\mathrm{ld}} \setminus \bigcup_{P\in\mathcal{P}} P$. We first bound the number of connected components.

▶ **Lemma 18.** *V_{ldmod} contains at most $12k$ connected components.*

Proof. A clique can intersect at most one connected component of V_{ldmod}. Since Reduction Rule 8 is exhaustively applied, V_{ld} can be partitioned into at most $8k$ cliques and $|S| \leq 4k$ vertices. Therefore, there can be at most $8k + 4k = 12k$ connected components in V_{ldmod}. ◀

We state that each connected component of V_{ldmod} has a specific structure. A vertex set Z is *extended clique-module* in a graph $G' = (V', E')$ if

(i) for each different $u, v \in Z$, $(u, v) \in E'$, and
(ii) for each $u, v \in Z$, $N_{G'}(u) \cup \{u\} = N_{G'}(v) \cup \{v\}$.

In other words, an extended clique-module is a *clique-module* [17] in the graph obtained by reducing the multiplicity of each multi-edge to 1. We can prove that each connected component of V_{ldmod} is actually an extended clique-module in V_{ld}.

▶ **Lemma 19.** *Each connected component of V_{ldmod} is an extended clique-module in V_{ld}.*

Proof. Let C be a connected component of V_{ldmod}. C should satisfy condition (i), because otherwise C would contain an induced P_3, contradicting the maximality of $\mathcal{P}$. Moreover, C should satisfy condition (ii), because otherwise there would exist $u \in \bigcup_{P\in\mathcal{P}} P$ and $v_1, v_2 \in C$ such that $(u, v_1) \in E$ and $(u, v_2) \notin E$, which would form an induced P_3, again contradicting the maximality of $\mathcal{P}$. ◀

Let E_{mul} be the set of multi-edges of G. Since any feasible solution should contain at least one endpoint of each edge in E_{mul}, the graph $G_{\mathrm{mul}} := (V, E_{\mathrm{mul}})$ should have a vertex cover of size k. This observation leads to the following two reduction rules called *Buss rule* [5, 7, 17] in the literature of kernelization of Vertex Cover, which are clearly safe.

▶ **Reduction Rule 11.** *If there is a vertex $v \in V$ with $|N_{G_{\mathrm{mul}}}(v)| > k$, remove v and decrease k by 1.*

▶ **Reduction Rule 12.** *If $|E_{\mathrm{mul}}| > k^2$, return* no.

We finish the analysis by bounding the number of vertices in each connected component of V_{ldmod} that is neither adjacent to a vertex outside V_{ld} nor incident to a multi-edge. We have the following.

▶ **Lemma 20.** *Let $u, v \in V_{\mathrm{ld}}$ with $(u, v) \in E$ and assume neither u nor v is incident to a multi-edge. Assume $N_G(u) \cup \{u\} = N_G(v) \cup \{v\}$. Then, for any minimum feasible solution X, either $\{u, v\} \subseteq X$ or $\{u, v\} \cap X = \emptyset$ holds.*

Proof. Assume a feasible solution X satisfies $u \in X$ and $v \notin X$. It is sufficient to prove that $X \setminus \{u\}$ is still feasible. Let C be a connected component of $G - (X \setminus \{u\})$ containing both u and v. Since X is feasible, all connected components of $G - (X \setminus \{u\})$ other than C are either cliques or trees. It suffices to show that C is a clique. Since $v \in V_{\mathrm{ld}}$, the connected component C' of $G - X$ that contains v is a clique. Since $N_{G-(X\setminus\{u\})}(u) = N_{G-(X\setminus\{u\})}(v) \cup \{v\} \setminus \{u\} = N_{G-X}(v) \cup \{v\} \setminus \{u\} = C'$, we obtain $C = C' \cup \{u\}$. Since there are no multi-edges between u and C', C is a clique. ◀

Now, we apply the following reduction rule.

▶ **Reduction Rule 13.** *Assume a connected component of V_{ldmod} contains $k+4$ vertices that are not adjacent to vertices outside V_{ld} and incident to no multi-edges. Then, remove one of those $k+4$ vertices.*

▶ **Lemma 21.** *Reduction Rule 13 is safe.*

Proof. Let Z be a set of vertices satisfying the assumption of the rule. Clearly, Z induces a clique. Since each vertex in Z has no neighbor outside V_{ld}, from Lemma 19, the set $N_G(v) \cup \{v\}$ is same for all $v \in Z$. Let X be a minimum feasible solution for G. Then, Lemma 20 ensures $Z \cap X = \emptyset$ because $|Z| > k$. Therefore, X is still a feasible solution for $G - v$ for any $v \in Z$. Conversely, let $v \in Z$ and X' be a feasible solution for $G - v$. Since $Z \setminus \{v\}$ is a clique of size $k+3$, $Z - v - X'$ is contained in a single clique component C' of $G - v - X'$. Moreover, we have $N_{G-X'}(v) = N_{G-X'}(u) \cup \{u\} \setminus \{v\}$ for all $u \in Z \setminus (X' \cup \{v\})$, and thus, $N_{G-X'}(v) = C'$. Particularly, $C' \cup \{v\}$ is a clique component of $G - X'$. Therefore, X' is still a feasible solution for G. ◀

Now we can bound $|V_{\mathrm{ld}}|$ by $O(k^2)$.

▶ **Lemma 22.** *After applying all the above reduction rules, we have $|V_{\mathrm{ld}}| \leq 101k^2 + 40k$.*

Proof. A vertex v is in V_{ld} only when

- **(i)** it has a neighbor outside V_{ld},
- **(ii)** it is in $\bigcup_{P \in \mathcal{P}} P$,
- **(iii)** there is a multi-edge incident to v, or
- **(iv)** it does not satisfy neither of condition (i), (ii), and (iii).

Lemma 15 states that at most $84k^2 + 4k$ vertices satisfy condition (i). Reduction Rule 10 ensures that at most $3k^2$ vertices satisfy condition (ii). Reduction Rule 12 ensures that at most $2k^2$ vertices satisfy condition (iii). Lemma 18 and Reduction Rule 13 ensures at most $12k \cdot (k+3) = 12k^2 + 36k$ vertices satisfy condition (iv). Therefore, we have $|V_{\mathrm{ld}}| \leq (84k^2 + 4k) + 3k^2 + 2k^2 + (12k^2 + 36k) \leq 101k^2 + 40k$. ◀

3.4 Bounding Number of Vertices

Now, we finish the overall analysis by bounding the number of vertices in $V \setminus V_{\mathrm{ld}}$. We first bound the number of edges between S and the tree components of $G - S$ (remember, S is a 4-approximate solution). Let $V_{\mathrm{tree}} := \bigcup_{C \in \mathcal{C}_{\mathrm{tree}}} C$. We begin by the following.

▶ **Reduction Rule 14.** *If there is a vertex $v \in V_{\mathrm{ld}} \cap S$ such that $N(v)$ contains at least $2k+4$ vertices from V_{tree}, remove v and decrease k by 1.*

▶ **Lemma 23 (*).** *Reduction Rule 14 is safe.*

Thus, we can bound the number of vertices in V_{tree} adjacent to some vertex in S.

▶ **Lemma 24 (*).** *After exhaustively applying all the above reduction rules, at most $28k^2$ vertices in V_{tree} are adjacent to some vertex in S.*

We apply the following reduction rules by local structures. Most of them appear in [28], but Reduction Rule 19 is new.

▶ **Reduction Rule 15** ([28])**.** *If a vertex v is adjacent to at least two pendant vertices, remove one of them.*

▶ **Reduction Rule 16** ([28]). *If there are multi-edges with a multiplicity of at least* 3*, reduce the multiplicity to* 2*.*

▶ **Reduction Rule 17** ([28]). *If there are three different vertices* (v_1, v_2, v_3) *such that* $(v_1, v_2), (v_2, v_3) \in E$*,* $(v_1, v_3) \notin E$*,* $d(v_2) = 2$*, and* $d(v_3) = 1$*, remove* v_3*.*

▶ **Reduction Rule 18** ([28]). *If there are five different vertices* $(v_1, v_2, v_3, v_4, v_5)$ *such that* (v_1, v_2)*,* (v_2, v_3)*,* (v_3, v_4)*,* (v_4, v_5) *are in* E *and* $d(v_2) = d(v_3) = d(v_4) = 2$*, remove* v_3 *and add an edge* (v_2, v_4)*.*

▶ **Reduction Rule 19.** *If there are four different vertices* (v_1, v_2, v_3, v_4) *such that* $(v_i, v_j) \in E$ *if and only if* $i = 1$*,* $d(v_1) = 3$*, and* $d(v_4) = 1$*, remove* v_4*.*

▶ **Lemma 25** (*). *Reduction Rule 19 is safe.*

Now, we bound $|V_{\mathrm{tree}}|$. We begin with the following.

▶ **Lemma 26** (*). *After applying all the above reduction rules,* V_{tree} *contains at most* $28k^2$ *pendant vertices that are adjacent to a vertex of degree* 3*.*

Now, we can prove the following.

▶ **Lemma 27** (*). *We have* $|V_{\mathrm{tree}}| \leq 1232k^2$*.*

Now, we can bound the size of the whole graph, which directly proves Theorem 1.

▶ **Lemma 28** (*). *After applying all the above reduction rules, we have* $|V| \leq 1389k^2 + 52k$*.*

References

1. Faisal Abu-Khzam. A kernelization algorithm for d-hitting set. *Journal of Computer and System Sciences*, 76(7):524–531, 2010. doi:10.1016/j.jcss.2009.09.002.
2. Matthias Bentert, Michael Fellows, Petr Golovach, Frances Rosamond, and Saket Saurabh. Breaking a graph into connected components with small dominating sets. In *International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 24–1, 2024. doi:10.4230/LIPIcs.MFCS.2024.24.
3. Benjamin Bergougnoux, Mamadou Moustapha Kanté, and O-joung Kwon. An optimal XP algorithm for hamiltonian cycle on graphs of bounded clique-width. *Algorithmica*, 82(6):1654–1674, 2020. doi:10.1007/s00453-019-00663-9.
4. Stéphane Bessy, Marin Bougeret, Dimitrios Thilikos, and Sebastian Wiederrecht. Kernelization for graph packing problems via rainbow matching. In *Symposium on Discrete Algorithms (SODA)*, pages 3654–3663. SIAM, 2023. doi:10.1137/1.9781611977554.ch139.
5. Jonathan Buss and Judy Goldsmith. Nondeterminism within P^*. *SIAM Journal on Computing*, 22(3):560–572, 1993. doi:10.1137/0222038.
6. Yixin Cao and Dániel Marx. Interval deletion is fixed-parameter tractable. *ACM Transactions on Algorithms (TALG)*, 11(3):1–35, 2015. doi:10.1145/2629595.
7. Marek Cygan, Fedor Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 5. Springer, 2015. doi:10.1007/978-3-319-21275-3.
8. Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Fast hamiltonicity checking via bases of perfect matchings. *Journal of the ACM (JACM)*, 65(3):1–46, 2018. doi:10.1145/3148227.
9. Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michał Pilipczuk, Johan MM Van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. *ACM Transactions on Algorithms (TALG)*, 18(2):1–31, 2022. doi:10.1145/3506707.

10 Martin Doucha and Jan Kratochvíl. Cluster vertex deletion: A parameterization between vertex cover and clique-width. In *International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 348–359. Springer, 2012. doi:10.1007/978-3-642-32589-2_32.

11 Rod Downey and Michael Fellows. Fixed-parameter tractability and completeness i: Basic results. *SIAM Journal on computing*, 24(4):873–921, 1995. doi:10.1137/S0097539792228228.

12 Michael Fellows, Lars Jaffke, Aliz Izabella Király, Frances Rosamond, and Mathias Weller. What is known about vertex cover kernelization? In *Adventures Between Lower Bounds and Higher Altitudes: Essays Dedicated to Juraj Hromkovič on the Occasion of His 60th Birthday*, pages 330–356. Springer, 2018. doi:10.1007/978-3-319-98355-4_19.

13 Fedor Fomin, Petr Golovach, Daniel Lokshtanov, and Saket Saurabh. Almost optimal lower bounds for problems parameterized by clique-width. *SIAM Journal on Computing*, 43(5):1541–1563, 2014. doi:10.1137/130910932.

14 Fedor Fomin, Petr Golovach, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. Clique-width III: Hamiltonian cycle and the odd case of graph coloring. *ACM Transactions on Algorithms (TALG)*, 15(1):1–27, 2018. doi:10.1145/3280824.

15 Fedor Fomin, Tien-Nam Le, Daniel Lokshtanov, Saket Saurabh, Stéphan Thomassé, and Meirav Zehavi. Subquadratic kernels for implicit 3-hitting set and 3-set packing problems. *ACM Transactions on Algorithms (TALG)*, 15(1):1–44, 2019. doi:10.1145/3293466.

16 Fedor Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Planar $\mathcal{F}$-deletion: Approximation, kernelization and optimal fpt algorithms. In *Annual Symposium on Foundations of Computer Science (FOCS)*, pages 470–479. IEEE, 2012. doi:10.1109/FOCS.2012.62.

17 Fedor Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: theory of parameterized preprocessing*. Cambridge University Press, 2019.

18 Esther Galby, Liana Khazaliya, Fionn Mc Inerney, Roohani Sharma, and Prafullkumar Tale. Metric dimension parameterized by feedback vertex set and other structural parameters. *SIAM Journal on Discrete Mathematics*, 37(4):2241–2264, 2023. doi:10.1137/22m1510911.

19 Robert Ganian. Improving vertex cover as a graph parameter. *Discrete Mathematics & Theoretical Computer Science*, 17(2):77–100, 2015. doi:10.46298/dmtcs.2136.

20 Tatsuya Gima, Eun Jung Kim, Noleen Köhler, Nikolaos Melissinos, and Manolis Vasilakis. Bandwidth parameterized by cluster vertex deletion number. In *International Symposium on Parameterized and Exact Computation (IPEC)*, volume 285, pages 21:1–21:15, 2023. doi:10.4230/LIPIcs.IPEC.2023.21.

21 Petr Golovach, R Krithika, Abhishek Sahu, Saket Saurabh, and Meirav Zehavi. Graph hamiltonicity parameterized by proper interval deletion set. In *Latin American Symposium on Theoretical Informatics*, pages 104–115. Springer, 2020. doi:10.1007/978-3-030-61792-9_9.

22 Jiong Guo, Christian Komusiewicz, Rolf Niedermeier, and Johannes Uhlmann. A more relaxed model for graph-based data clustering: s-plex cluster editing. *SIAM Journal on Discrete Mathematics*, 24(4):1662–1683, 2010. doi:10.1137/090767285.

23 David Harris and N. S. Narayanaswamy. A faster algorithm for vertex cover parameterized by solution size. In *International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 289, pages 40:1–40:18, 2024. doi:10.4230/LIPIcs.STACS.2024.40.

24 Falk Hüffner, Christian Komusiewicz, Hannes Moser, and Rolf Niedermeier. Fixed-parameter algorithms for cluster vertex deletion. *Theory of Computing Systems*, 47(1):196–217, 2010. doi:10.1007/s00224-008-9150-x.

25 Yoichi Iwata. Linear-time kernelization for feedback vertex set. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 80 of *LIPIcs*, pages 68:1–68:14, 2017. doi:10.4230/LIPIcs.ICALP.2017.68.

26 Ashwin Jacob, Jari JH de Kroon, Diptapriyo Majumdar, and Venkatesh Raman. Deletion to scattered graph classes I-case of finite number of graph classes. *Journal of Computer and System Sciences*, 138:103460, 2023. doi:10.1016/j.jcss.2023.05.005.

27 Ashwin Jacob, Diptapriyo Majumdar, and Venkatesh Raman. Deletion to scattered graph classes II-improved FPT algorithms for deletion to pairs of graph classes. *Journal of Computer and System Sciences*, 136:280–301, 2023. doi:10.1016/j.jcss.2023.03.004.

28 Ashwin Jacob, Diptapriyo Majumdar, and Meirav Zehavi. A polynomial kernel for deletion to the scattered class of cliques and trees. In *International Symposium on Algorithms and Computation (ISAAC)*, volume 322 of *LIPIcs*, pages 41:1–41:17, 2024. doi:10.4230/LIPIcs.ISAAC.2024.41.

29 Ashwin Jacob, Michal Wlodarczyk, and Meirav Zehavi. Finding long directed cycles is hard even when DFVS is small or girth is large. In *European Symposium on Algorithms (ESA)*, volume 274, pages 65:1–65:17, 2023. doi:10.4230/LIPIcs.ESA.2023.65.

30 Bart Jansen, Jari de Kroon, and Michal Wlodarczyk. Single-exponential FPT algorithms for enumerating secluded $\mathcal{F}$-free subgraphs and deleting to scattered graph classes. *Journal of Computer and System Sciences*, 148:103597, 2025. doi:10.1016/j.jcss.2024.103597.

31 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 Transactions on Algorithms (TALG)*, 12(2):1–41, 2015. doi:10.1145/2797140.

32 Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 450–459. IEEE, 2012. doi:10.1109/FOCS.2012.46.

33 Michael Lampis, Georgia Kaouri, and Valia Mitsou. On the algorithmic effectiveness of digraph decompositions and complexity measures. *Discrete Optimization*, 8(1):129–138, 2011. doi:10.1016/j.disopt.2010.03.010.

34 Jason Li and Jesper Nederlof. Detecting feedback vertex sets of size k in $O^*(2.7^k)$ time. *ACM Transactions on Algorithms (TALG)*, 18(4):34:1–34:26, 2022. doi:10.1145/3504027.

35 Hong Liu, Peng Zhang, and Daming Zhu. On editing graphs into 2-club clusters. In *Frontiers in Algorithmics and Algorithmic Aspects in Information and Management (FAW-AAIM)*, pages 235–246. Springer, 2012. doi:10.1007/978-3-642-29700-7_22.

36 Dániel Marx. Chordal deletion is fixed-parameter tractable. *Algorithmica*, 57:747–768, 2010. doi:10.1007/s00453-008-9233-8.

37 Ashutosh Rai and Saket Saurabh. Bivariate complexity analysis of almost forest deletion. *Theoretical Computer Science*, 708:18–33, 2018. doi:10.1016/j.tcs.2017.10.021.

38 Bruce Reed, Kaleigh Smith, and Adrian Vetta. Finding odd cycle transversals. *Operations Research Letters*, 32(4):299–301, 2004. doi:10.1016/j.orl.2003.10.009.

39 Stéphan Thomassé. A $4k^2$ kernel for feedback vertex set. *ACM Transactions on Algorithms (TALG)*, 6(2):32:1–32:8, 2010. doi:10.1145/1721837.1721848.

40 Dekel Tsur. Faster algorithms and a smaller kernel for cliques or trees vertex deletion. *Information Processing Letters*, 190:106570, 2025. doi:10.1016/j.ipl.2025.106570.

41 Meirav Zehavi. Tournament fixing parameterized by feedback vertex set number is FPT. In *AAAI Conference on Artificial Intelligence*, pages 5876–5883, 2023. doi:10.1609/aaai.v37i5.25728.

Maximizing Social Welfare Among EF1 Allocations at the Presence of Two Types of Agents

Jiaxuan Ma
Hangzhou Dianzi University, China

Yong Chen
Hangzhou Dianzi University, China

Guangting Chen
Zhejiang University of Water Resources and Electric Power, Hangzhou, China

Mingyang Gong
University of Alberta, Edmonton, Canada

Guohui Lin
University of Alberta, Edmonton, Canada

An Zhang
Hangzhou Dianzi University, China

Abstract

We study the fair allocation of indivisible items to n agents to maximize the utilitarian social welfare, where the fairness criterion is envy-free up to one item and there are only two different utility functions shared by the agents. We present a 2-approximation algorithm when the two utility functions are normalized, improving the previous best ratio of $16\sqrt{n}$ shown for general normalized utility functions; thus this constant ratio approximation algorithm confirms the APX-completeness in this special case previously shown APX-hard. When there are only three agents, i.e., $n = 3$, the previous best ratio is 3 shown for general utility functions, and we present an improved and tight $\frac{5}{3}$-approximation algorithm when the two utility functions are normalized, and a best possible and tight 2-approximation algorithm when the two utility functions are unnormalized.

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

Keywords and phrases Fair allocation, utilitarian social welfare, envy-free up to one item, envy-cycle elimination, round robin, approximation algorithm

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.49

Related Version *Full Version*: https://doi.org/10.48550/arXiv.2509.09641

Funding Supported by the NSERC Canada, the NNSF of China (Grants 12471301, 12371316), the PNSF of Zhejiang China (Grant LZ25A010001), and the China Scholarship Council (Grant No. 202508330173).

1 Introduction

We study a special case of the fair allocation of indivisible items to a number of agents to maximize their utilitarian social welfare, where the agents fall into two types and the fairness criterion is envy-free up to one item. We present a set of approximation algorithms that each improves the previous best one when reduced to this special case, or is the best possible for this special case.

Throughout this paper, we denote by $\{a_1, a_2, \ldots, a_n\}$ the set of n agents and by $M = \{g_1, g_2, \ldots, g_m\}$ the set of m indivisible items or goods. An *allocation* of items is an n-tuple $\mathcal{A} = (A_1, A_2, \ldots, A_n)$, where A_i is the pair-wise disjoint bundle/subset of items allocated/assigned to a_i. The allocation is *complete* if $\cup_{i=1}^{n} A_i = M$, i.e., every item is

© Jiaxuan Ma, Yong Chen, Guangting Chen, Mingyang Gong, Guohui Lin, and An Zhang;
licensed under Creative Commons License CC-BY 4.0

allocated, or otherwise it is *partial*. Each agent a_i has a non-negative *additive utility* (or *valuation*) function $u_i : M \to \mathbb{R}_{\geq 0}$. Extending to 2^M, for any subset $S \subseteq M$, $u_i(S) = \sum_{g \in S} u_i(g)$. If $u_i(M) = 1$ for all i, then the utility functions are said *normalized*; otherwise, they are *unnormalized*. The *utilitarian social welfare* of the allocation $\mathcal{A} = (A_1, A_2, \ldots, A_n)$ is defined as $\sum_{i=1}^{n} u_i(A_i)$.

The allocation $\mathcal{A} = (A_1, A_2, \ldots, A_n)$ is *envy-free up to one item* (EF1), if for any $i \neq j$, $u_i(A_i) \geq u_i(A_j \setminus g)$ holds for some item $g \in A_j$. Between two agents a_i and a_j, if $u_i(A_i) \geq u_i(A_j)$, then a_i does not envy a_j; otherwise, a_i envies a_j. Similarly, if $u_i(A_i) \geq u_i(A_j \setminus g)$ for some $g \in A_j$, then a_i does not *strongly* envy a_j; otherwise, a_i strongly envies a_j.

In this paper, we study the optimization problem to find an EF1 allocation with the maximum utilitarian social welfare, denoted as USW-EF1, and we focus on the special case where there are only two types of agents, or equivalently speaking only two distinct utility functions shared by the agents. W.l.o.g., we assume that the first j agents $a_1, a_2, \ldots, a_j$ use the utility function u_1, referred to as the first type of agents, and the last $n - j$ agents $a_{j+1}, a_{j+2}, \ldots, a_n$ use the second utility function u_n, for some $1 \leq j < n$. Let $\mathcal{A}$ be the allocation produced by an approximation algorithm and $\mathcal{A}^*$ be an optimal allocation. We denote by $SOL = \sum_{i=1}^{n} u_i(A_i)$ and $OPT = \sum_{i=1}^{n} u_i(A_i^*)$ their objective values, respectively. The algorithm is an α-approximation if the ratio $OPT/SOL \leq \alpha$ for all instances. We remark that this two-types-of-agents special case is important for several reasons, for example, most hardness and inapproximability results are proven for this special case, and this special case also models some real application scenarios such as individual versus business investors and faculty members versus administration staff.

1.1 Related work

Discrete fair division aims to allocate a set of indivisible items to a set of agents so that all the agents feel "fair" with respect to their heterogeneous preferences over the items. Typical fairness criteria include *envy-freeness* (EF) [12] and *proportionality* [11]. By an EF allocation, each agent a_i does not envy any other agent a_j. Since an EF allocation does not always exist, one relaxation of EF is envy-freeness up to one item (EF1) proposed in [9, 7]. Formally speaking, for each pair of agents a_i, a_j, agent a_i does not envy a_j after the removal of *some* item from the bundle allocated to a_j. Replacing the quantifier "for some" by "for any" leads to another more restricted relaxation *envy-freeness up to any item* (EFX) [8].

A complete EF1 allocation can be computed in polynomial time by the *envy-cycle elimination* (ECE) algorithm proposed by Lipton et al. [9] or by the *round-robin* (RR) algorithm proposed by Caragiannis et al. [8]. In brief, in the ECE algorithm, an *envy digraph* is constructed on the agents where agent a_i *points* to agent a_j (i.e., the arc (a_i, a_j) exists) if a_i envies a_j; whenever there is a cycle in the digraph, the bundles allocated to the agents on the cycle are rotated to eliminate the cycle; afterwards the algorithm assigns an unallocated item to an agent not envied by anyone else, and the iteration goes on. In the RR algorithm, the agents are ordered cyclically and the agent at the head of the order picks its most preferred item from the pool of unassigned items and then lines up at the back of the order, until no item is left.

Besides the existence of EF1 allocations, the price of such fair allocations is an interesting concern in economics and social science. Given that the utilitarian social welfare measures the *efficiency* of an allocation, the *price of EF1* is defined as the ratio between the maximum utilitarian social welfare across all allocations and the maximum utilitarian social welfare among only EF1 allocations. The price of EF1 for normalized utility functions is $\Theta(\sqrt{n})$: Barman et al. [4] first showed that the price is at most $O(\sqrt{n})$, and later Bei et al. [5] showed

that the price is at least $\Omega(\sqrt{n})$. The price of EF1 for unnormalized utility functions is n: Barman et al. [4] showed that the price is $\Theta(n)$, and later Bu et al. [6] constructed an instance to show that it is at least n.

For the optimization problem USW-EF1, Barman et al. [4] proved that it is already NP-hard for two agents even when the valuation function of an agent is a scaled version of that of the other, and it is NP-hard to approximate within a factor of $m^{1-\epsilon}$ for any $\epsilon > 0$ for n agents and m items when both n and m are part of the input. When the utility functions are normalized, the problem remains NP-hard for n agents when $n \geq 2$ is a fixed integer [3, 6]. Moreover, Bu et al. [6] showed several hardness results when $n \geq 3$ that, firstly the problem is NP-hard to approximate within the factor $\frac{4n}{3n+1}$ for normalized functions when one agent uses a utility function and the other agents use a common second utility function; secondly it is NP-hard to approximate within the factor $\frac{1+\sqrt{4n-3}}{2}$ for unnormalized functions when one agent uses a utility function, two other agents use a common second utility function, and all the other agents (if any) use a common third utility function; and thirdly it is NP-hard to approximate within the factor $m^{\frac{1}{2}-\epsilon}$ or within the factor $n^{\frac{1}{3}-\epsilon}$, for any $\epsilon > 0$, for normalized utility functions when both n and m are part of the input.

On the positive side, Barman et al. [4] presented a $16\sqrt{n}$-approximation algorithm when n is a fixed integer and the utility functions are normalized. Aziz et al. [3] proposed a pseudo-polynomial time exact algorithm for the same variant. Bu et al. [6] gave a *fully polynomial-time approximation scheme* (FPTAS) for two agents and an n-approximation algorithm for $n \geq 3$ agents with unnormalized utility functions.

We remark that we reviewed in the above only those directly related work, but not the entire body of the literature on fair division of indivisible goods. For example, the Nash social welfare (NSW) objective, the geometric mean of the agents' utilities, has been extensively studied, and it is known that an allocation which maximizes NSW is EF1 [8] though approximate solutions are not necessarily EF1. One may refer to [2, 1] for an excellent survey on recent progress and open questions.

1.2 Our contribution and organization of the paper

In this paper, we aim to design approximation algorithms for the special case of USW-EF1 where there are only two distinct utility functions shared by the agents, i.e., there are two types of agents. Note that when all agents share the same utility function, a complete EF1 allocation returned by the ECE or RR algorithm is optimal. Also, given that the above two lower bounds on the approximation ratios [6] are proven for two or three utility functions, our study on the special case may shed lights on the general case. In particular, we demonstrate the use of the *item preference order* in all our three approximation algorithms.

We first present in the next section a 2-approximation algorithm for any number of agents with normalized utility functions. Then, in Section 3, we present a tight 2-approximation algorithm for three agents with unnormalized utility functions, which is the *best possible* by the lower bound $\frac{1+\sqrt{4n-3}}{2}$ [6]. Section 4 contains an improved and tight $\frac{5}{3}$-approximation algorithm for three agents with normalized utility functions. Due to page limit, we leave the third case for three agents with unnormalized utility functions to the full arXiv version [10]. We conclude our paper in Section 5.

In all our three algorithms, the items are firstly sorted, and thus we assume w.l.o.g. that they are given, in the non-increasing order of their *preferences*, where the preference of an item g is defined as $u_1(g)/u_n(g)$ (Definition 1) that measures the extent of preference of the first type of agents over the second type of agents. Intuitively, in an efficient allocation, items at the front of this order should be allocated to the first type of agents, while items at the back should be allocated to the second type of agents. This is exactly the main design idea.

2 A 2-approximation for normalized functions

Recall that agents $a_1, \ldots, a_j$ share the first utility function $u_1(\cdot)$ and agents $a_{j+1}, \ldots, a_n$ share the second utility function $u_n(\cdot)$. For each item $g \in M$, we assume w.l.o.g. that its two values $u_1(g), u_n(g)$ are not both 0.

▶ **Definition 1.** *The* preference *of item g is defined as $\rho(g) = u_1(g)/u_n(g)$, where if $u_n(g) = 0$ then $\rho(g) = \infty$.*

Extending to a non-empty set of items $S \subseteq M$, the preference of S is defined as $\rho(S) = u_1(S)/u_n(S)$, where if $u_n(S) = 0$ then $\rho(S) = \infty$.

Specially, the preference of an empty set is $\pm\infty$, indicating that it is less than but also greater than any real value.

We assume w.l.o.g. that $\rho(g_1) \geq \rho(g_2) \geq \ldots \geq \rho(g_m)$, i.e., the items are given in the *non-decreasing preference* order. Our algorithm, denoted as Approx1, uses two variables k_1 and k_2 to store the smallest and the largest indices of the unassigned items, which are initialized to 1 and m, respectively. In each iteration, the algorithm finds the agent a_s (a_t, resp.) having the minimum utility among the first (second, resp.) type of agents. (*Comment: These two agents a_s and a_t will be proven not to envy each other.*) If a_s is not envied by a_t, then a_s takes the item g_{k_1} and k_1 is incremented by 1; otherwise, a_t is not envied by a_s, a_t takes g_{k_2} and k_2 is decremented by 1. The algorithm terminates when all items are assigned (i.e., $k_1 > k_2$), of which a high-level description is depicted in Algorithm 1.

Algorithm 1 Approx1 for normalized utility functions.

Input: $n \geq 3$ agents of two types and a set of m indivisible items $\rho(g_1) \geq \rho(g_2) \geq \ldots \geq \rho(g_m)$.
Output: A complete EF1 allocation.

```
Initialize k_1 = 1, k_2 = m, and A_i = ∅ for every agent a_i;
while (k_1 ≤ k_2) do
    find s = arg min_{i=1}^{j} u_1(A_i) and t = arg min_{i=j+1}^{n} u_n(A_i);
    if (a_s is not envied by a_t) then
        A_s = A_s ∪ {g_{k_1}} and k_1 = k_1 + 1;
    else
        A_t = A_t ∪ {g_{k_2}} and k_2 = k_2 − 1;
return the final allocation.
```

We next prove that inside each iteration, the found two agents a_s and a_t do not envy each other, and thus Approx1 produces a complete EF1 allocation. To this purpose, we introduce the following definition.

▶ **Definition 2.** *For two item sets $A, B \subseteq M$, if $\min_{g \in A \setminus B} \rho(g) \geq \max_{g \in B \setminus A} \rho(g)$, then we say A* precedes *B and denote it as $A \prec B$. That is, excluding the common items, every item of A (if any) comes before any item of B (if any) in the item preference order.*

An allocation $\mathcal{A}$ is good *if $A_s \prec A_t$ for every pair (s, t) with $s \leq j < t$. (*Comment: Since $A_s \cap A_t = \emptyset$, this implies $\rho(A_s) \geq \rho(A_t)$.*)*

Recall that $\rho(\emptyset) = \pm\infty$, if $A_s = \emptyset$ then both $A_s \prec A_t$ and $A_t \prec A_s$ hold.

▶ **Lemma 3.** *Given a good allocation $\mathcal{A}$, two agents of different types do not mutually envy each other.*

Proof. Assume to the contrary that for a pair (s, t) with $s \leq j < t$, a_s and a_t envy each other, that is, $u_1(A_s) < u_1(A_t)$ and $u_n(A_t) < u_n(A_s)$. It follows that none of A_s and A_t is $\emptyset$ and $\rho(A_s) = u_1(A_s)/u_n(A_s) < u_1(A_t)/u_n(A_t) = \rho(A_t)$, which contradicts the definition of a good allocation. ◀

Note that there are exactly m iterations inside the algorithm Approx1, and we let $\mathcal{A}^0 = (\emptyset, \ldots, \emptyset)$ and let $\mathcal{A}^i = (A_1^i, \ldots, A_n^i)$ denote the allocation at the *end* of the i-th iteration. The final produced allocation is $\mathcal{A} = \mathcal{A}^m$.

▶ **Lemma 4.** *The allocation $\mathcal{A}^i$ is good and EF1, for each $i = 0, 1, \ldots, m$.*

Proof. We prove by induction.

The initial empty allocation $\mathcal{A}^0$ is clearly good and EF1. Assume $\mathcal{A}^i$ is good and EF1 for some $i < m$, and let a_s and a_t be the agents found in the $(i+1)$-st iteration.

Consider the case where a_s is not envied by a_t (the other case is symmetric), in which the algorithm updates A_s to $A_s \cup \{g_{k_1}\}$. By the description of Approx1, $A_s \cup \{g_{k_1}\}$ is a non-empty subset of $\{g_1, \ldots, g_{k_1}\}$ and $A_i \subseteq \{g_{k_2+1}, \ldots, g_m\}$ for every $i \geq j+1$. By Definitions 1 and 2 and $k_1 \leq k_2$, $A_s \cup \{g_{k_1}\} \prec A_i$ for every $i \geq j+1$, and thus $\mathcal{A}^{i+1}$ is good.

Since at the beginning of the iteration a_s is not envied by a_t, $u_n(A_t) \geq u_n(A_s)$. Also, by the definition of s and t, $u_1(A_i) \geq u_1(A_s)$ for every $i \leq j$ and $u_n(A_i) \geq u_n(A_t) \geq u_n(A_s)$ for every $i \geq j+1$. That is, no agent envies a_s in the allocation $\mathcal{A}^i$, and thus does not strongly envy a_s in the allocation $\mathcal{A}^{i+1}$ (by removing the item g_{k_1}). Note that in this iteration only a_s gets an item. Therefore, $\mathcal{A}^{i+1}$ is EF1 as $\mathcal{A}^i$ is EF1. ◀

▶ **Theorem 5.** Approx1 *is a 2-approximation algorithm.*

Proof. For the optimal allocation $\mathcal{A}^*$, we have

$$OPT = \sum_{i=1}^{n} u_i(A_i^*) = u_1(\cup_{i=1}^{j} A_i^*) + u_n(\cup_{i=j+1}^{n} A_i^*) \leq u_1(M) + u_n(M) = 2.$$

Note from Lemma 4 that the final allocation $\mathcal{A}$ produced by Approx1 is complete, good and EF1. By Definition 2, we have $\rho(\cup_{i=1}^{j} A_i) \geq \rho(\cup_{i=j+1}^{n} A_i)$. Therefore, either $\rho(\cup_{i=1}^{j} A_i) \geq 1$ or $\rho(\cup_{i=j+1}^{n} A_i) < 1$; or equivalently, either $u_1(\cup_{i=1}^{j} A_i) \geq u_n(\cup_{i=1}^{j} A_i)$ or $u_1(\cup_{i=j+1}^{n} A_i) < u_n(\cup_{i=j+1}^{n} A_i)$. In the first case, we have

$$SOL = \sum_{i=1}^{n} u_i(A_i) = u_1(\cup_{i=1}^{j} A_i) + u_n(\cup_{i=j+1}^{n} A_i) = u_1(\cup_{i=1}^{j} A_i) - u_n(\cup_{i=1}^{j} A_i) + u_n(M) \geq 1.$$

In the second case, we have

$$SOL = u_1(\cup_{i=1}^{j} A_i) + u_n(\cup_{i=j+1}^{n} A_i) = u_1(M) - u_1(\cup_{i=j+1}^{n} A_i) + u_n(\cup_{i=j+1}^{n} A_i) > 1.$$

Therefore, $SOL \geq \frac{1}{2} OPT$ always holds. ◀

3 A 2-approximation for three agents with unnormalized functions

We continue to assume w.l.o.g. that the items are given in their preference order, that is, $\rho(g_1) \geq \rho(g_2) \geq \ldots \geq \rho(g_m)$, and use the definitions of *precedence* and *good allocation* in Definition 2. Since there are only three agents categorized into two types, we assume w.l.o.g. that $j = 1$, i.e., agent a_1 is of the first type and agents a_2 and a_3 are of the second type.

For ease of presentation we partition the items into two sets based on their preference:

$$X = \{g \in M : u_1(g) \geq u_3(g)\}, \quad Y = \{g \in M : u_1(g) < u_3(g)\}. \tag{1}$$

We first examine the structure of a fixed optimal EF1 allocation, denoted as (A_1^*, A_2^*, A_3^*). Denote the item $g^* = \arg\max_{g \in A_1^*} u_3(g)$, i.e., the most valuable to a_2 and a_3 among those items allocated to a_1. The following lemma establishes an important upper bound on $u_3(A_1^* \setminus g^*)$.

▶ **Lemma 6.** $u_3(A_1^* \setminus g^*) \leq \frac{1}{3}u_3(M \setminus g^*)$.

Proof. Suppose to the contrary that $u_3(A_1^* \setminus g^*) > \frac{1}{3}u_3(M \setminus g^*)$. Then we have

$$u_3(A_2^* \cup A_3^*) = u_3(M) - u_3(A_1^*) = u_3(M \setminus g^*) - u_3(A_1^* \setminus g^*) < 2u_3(A_1^* \setminus g^*).$$

It follows that at least one of $u_3(A_2^*)$ and $u_3(A_3^*)$ is less than $u_3(A_1^* \setminus g^*)$. W.l.o.g., we assume $u_3(A_2^*) < u_3(A_1^* \setminus g^*)$, and thus by the definition of g^* agent a_2 strongly envies a_1, a contradiction to EF1. ◀

We next present an upper bound on the optimal total utility OPT. To this purpose, we define what a *critical set* is below. We remark that although the definition relies on the fixed optimal EF1 allocation $\mathcal{A}^*$, we can actually compute a critical set for any item, by the procedure Algo2 presented in Subsection 3.1.

▶ **Definition 7.** *An item set K is* critical for an item g *if the following three conditions are satisfied:*

1. $g \in K$ *and* $K \setminus g \subseteq X$*;*
2. $K \setminus g \prec A_1^* \setminus g$*;*
3. $u_3(K \setminus g) \geq \min\{u_3(X \setminus g), \frac{1}{3}u_3(M \setminus g)\}$.

▶ **Lemma 8.** *For any critical set K for g^*, we have $OPT \leq u_1(K) + u_3(M \setminus K)$.*

Proof. We partition the items of $A_1^* \setminus g^*$ into two sets according to Eq. (1):

$$B_1 = (A_1^* \setminus g^*) \cap X, \quad B_2 = (A_1^* \setminus g^*) \cap Y.$$

One sees from $B_2 \subseteq Y$ that $u_1(B_2) \leq u_3(B_2)$. Since $K \setminus g^* \prec A_1^* \setminus g^*$ and $B_1 \subseteq A_1^* \setminus g^*$, $K \setminus g^* \prec B_1$ too. Let $C = (K \setminus g^*) \cap B_1$; then $\rho(K \setminus \{g^* \cup C\}) \geq \rho(B_1 \setminus C)$. Since $K \setminus g^* \subseteq X$, $\rho(K \setminus g^*) \geq 1$. In summary, we have

$$\rho(K \setminus \{g^* \cup C\}) \geq \max\{\rho(B_1 \setminus C), 1\}.$$

By the third condition in Definition 7 and Lemma 6, we have

$$\begin{aligned} u_3(K \setminus \{g^* \cup C\}) &\geq \min\{u_3(X \setminus g^*), \frac{1}{3}u_3(M \setminus g^*)\} - u_3(C) \\ &\geq \min\{u_3(X \setminus g^*), u_3(A_1^* \setminus g^*)\} - u_3(C) \\ &\geq u_3(B_1 \setminus C). \end{aligned}$$

The above inequalities together give rise to

$$\begin{aligned} u_1(A_1^* \setminus g^*) - u_3(A_1^* \setminus g^*) &\leq u_1(B_1) - u_3(B_1) \\ &= u_1(C) - u_3(C) + u_1(B_1 \setminus C) - u_3(B_1 \setminus C) \\ &= u_1(C) - u_3(C) + u_3(B_1 \setminus C)(\rho(B_1 \setminus C) - 1) \\ &\leq u_1(C) - u_3(C) + u_3(K \setminus \{g^* \cup C\})(\rho(K \setminus \{g^* \cup C\}) - 1) \\ &= u_1(K \setminus g^*) - u_3(K \setminus g^*). \end{aligned} \tag{2}$$

It follows from the above Eq. (2) that

$$\begin{aligned} OPT &= u_1(g^*) + u_1(A_1^* \setminus g^*) + u_3(A_2^* \cup A_3^*) \\ &\leq u_1(g^*) + u_3(A_1^* \setminus g^*) + u_1(K \setminus g^*) - u_3(K \setminus g^*) + u_3(A_2^* \cup A_3^*) \\ &= u_1(K) + u_3(M \setminus K). \end{aligned}$$

This proves the lemma. ◀

The next lemma states an important property for any allocation in which agent a_1 is envied, by the fact that a_2 and a_3 share the same utility function u_3.

▶ **Lemma 9.** *Suppose $\mathcal{A}$ is an allocation in which agent a_1 is envied. If a_2 (a_3, resp.) is not envied by a_3 (a_2, resp.), then a_2 envies a_1, i.e., $u_3(A_2) < u_3(A_1)$ (a_3 envies a_1, i.e., $u_3(A_3) < u_3(A_1)$, resp.).*

Proof. We prove the lemma for a_2 (for a_3 it is symmetric), that is, agent a_2 is not envied by a_3, but a_1 is envied by at least one of a_2 and a_3.

If a_2 does not envy a_1, then a_3 envies a_1, i.e., $u_3(A_2) \geq u_3(A_1) > u_3(A_3)$, a contradiction to a_2 not envied by a_3. ◀

3.1 Producing a critical set for every item

Recall that the items are given in the preference order $\rho(g_1) \geq \rho(g_2) \geq \ldots \geq \rho(g_m)$; and we see from Eq. (1) that $X \prec Y$. For any ordered item set, if the order inherits the given preference order, then it is *regular*.

Note from Lemma 8 that a critical set for the item g^* plays an important role, but we have no knowledge of A_1^* or g^*. Below we construct a critical set for every item $g_i \in M$, in the procedure Algo2 depicted in Algorithm 2. The key idea is, excluding g_i, the critical set matches a prefix of the item preference order.

Algorithm 2 Algo2 for computing a critical set.

Input: The item preference order $S = \langle g_1, \ldots, g_m \rangle$ and an item $g_i \in M$.
Output: A critical set K for g_i.

1: Find the smallest t such that $\sum_{j=1}^{t} u_3(g_j) \geq \min\{u_3(X \setminus g_i), \frac{1}{3}u_3(M \setminus g_i)\}$;
▷ If $X \setminus g_i = \emptyset$, then $t = 0$.
2: **if** $(t < i)$ **then**
3: output $K = \{g_1, \ldots, g_t, g_i\}$;
4: **else**
5: find the smallest t' such that $\sum_{j=1}^{t'} u_3(g_j) \geq \min\{u_3(X), \frac{1}{3}u_3(M \setminus g_i) + u_3(g_i)\}$;
6: output $K = \{g_1, \ldots, g_{t'}\}$.

▶ **Lemma 10.** Algo2 *produces a critical set K for the item g_i.*

Proof. Consider the index t in Line 1 of the procedure Algo2. Clearly, $t \leq |X|$. If $t < i$, then $g_i \in K$ and $K \setminus g_i = \{g_1, \ldots, g_t\}$. Therefore, $K \setminus g_i \subseteq X$ and $K \setminus g_i \prec A_1^* \setminus g_i$. Lastly, $u_3(K \setminus g_i) = \sum_{j=1}^{t} u_3(g_j) \geq \min\{u_3(X \setminus g_i), \frac{1}{3}u_3(M \setminus g_i)\}$. All the three conditions in Definition 7 are satisfied and thus K is a critical set for g_i.

If $t \geq i$, then t' in Line 5 exists and $|X| \geq t' \geq t \geq i$. Therefore, $g_i \in K$, $K \subseteq X$ and $K \setminus g_i \prec A_1^* \setminus g_i$. Lastly, $u_3(K \setminus g_i) = \sum_{j=1}^{t'} u_3(g_j) - u_3(g_i) \geq \min\{u_3(X \setminus g_i), \frac{1}{3}u_3(M \setminus g_i)\}$. All the three conditions in Definition 7 are satisfied and thus K is a critical set for g_i. ◀

Given that the procedure Algo2 is able to produce a critical set for any item, by enumerating the items we will have a critical set K for the item g^*, and thus by Lemma 8 to bound the OPT. From K, we will also design algorithms to return a complete EF1 allocation of total utility at least half of this bound and thus at least $\frac{1}{2} \cdot OPT$.

In the next three subsections, we deal with the critical set K produced by Algo2 for an item g_i, separately for three possible cases. Let $k = |K|$. When $\max_{g \in K} u_1(g) < \frac{1}{2}u_1(K)$, K contains at least three items, i.e., $k \geq 3$; if the index of the item g_i is $i \geq k$, then we modify

the item order S by moving the item g_i to the $(k-1)$-st position, and denote the new order as $S' = \langle g'_1, \ldots, g'_m \rangle$. Note that the net effect is, if $i \geq k$, then the items $g_{k-1}, \ldots, g_{i-1}$ are moved one position forward to become $g'_k, \ldots, g'_i$; otherwise S' is identical to S. In either case, $S' \setminus g'_{k-1}$ is regular and K contains the first k items in the order S', summarized in Remark 11.

The three distinguished cases are (due to space limit, Case 3 is dealt with in detail in the full arXiv version [10]):

1. $\max_{g \in K} u_1(g) \geq \frac{1}{2} u_1(K)$;
2. $\max_{g \in K} u_1(g) < \frac{1}{2} u_1(K)$ and $u_3(g'_k) > u_3(M \setminus K)$;
3. $\max_{g \in K} u_1(g) < \frac{1}{2} u_1(K)$ and $u_3(g'_k) \leq u_3(M \setminus K)$.

▶ Remark 11. In Cases 2 and 3, i.e., $\max_{g \in K} u_1(g) < \frac{1}{2} u_1(K)$ and thus $k = |K| \geq 3$, for the new item order S', $g'_k \neq g_i$, $S' \setminus g'_{k-1}$ is regular, $K = \{g'_1, g'_2, \ldots, g'_k\}$, and $\sum_{j=1}^{k-1} u_3(g'_j) < \frac{1}{3} u_3(M \setminus g_i) + u_3(g_i)$.

Proof. We only need to prove the last inequality $\sum_{j=1}^{k-1} u_3(g'_j) < \frac{1}{3} u_3(M \setminus g_i) + u_3(g_i)$.

If K is outputted in Line 3 of the algorithm Algo2, then $k = t+1$ and $\sum_{j=1}^{k-1} u_3(g'_j) = \sum_{j=1}^{t-1} u_3(g_j) + u_3(g_i) < \frac{1}{3} u_3(M \setminus g_i) + u_3(g_i)$, i.e., the inequality holds. If K is outputted in Line 6 of the algorithm Algo2 and $i < t' = k$, then $\sum_{j=1}^{k-1} u_3(g'_j) = \sum_{j=1}^{t'-1} u_3(g_j) < \frac{1}{3} u_3(M \setminus g_i) + u_3(g_i)$, i.e., the inequality holds. If K is outputted in Line 6 of the algorithm Algo2 and $i = t = t' = k$, then $\sum_{j=1}^{k-1} u_3(g'_j) \leq \sum_{j=1}^{t-1} u_3(g_j) + u_3(g_i) < \frac{1}{3} u_3(M \setminus g_i) + u_3(g_i)$, i.e., the inequality holds. This proves the last inequality. ◀

3.2 Case 1: $\max_{g \in K} u_1(g) \geq \frac{1}{2} u_1(K)$

Recall that K is the critical set for the item g_i produced by the procedure Algo2. This case where $\max_{g \in K} u_1(g) \geq \frac{1}{2} u_1(K)$ is considered easy, and we present an algorithm Approx3 to produce a complete EF1 allocation.

The algorithm starts with the allocation $\mathcal{A} = (\{g\}, \emptyset, \emptyset)$, where the item g is one such that $u_1(g) \geq \frac{1}{2} u_1(K)$. $\mathcal{A}$ is trivially EF1 and $u_1(A_1) \geq \frac{1}{2} u_1(K)$. In fact, $\mathcal{A}$ is *well-defined* with permutation $(1, 2, 3)$, formally defined below.

▶ **Definition 12.** *An allocation $\mathcal{A}$ is* well-defined *with permutation (i_1, i_2, i_3) if*

1. *$\mathcal{A}$ is EF1;*
2. *$u_1(A_1) \geq \frac{1}{2} u_1(K)$ and $u_3(A_1) \leq u_3(K)$;*
3. *$(i_1, i_2, i_3) \in \{(1, 2, 3), (3, 1, 2)\}$ such that agent a_{i_1} does not envy a_{i_2} or a_{i_3}, and a_{i_2} does not envy a_{i_3}.*

Given the well-defined allocation $\mathcal{A} = (\{g\}, \emptyset, \emptyset)$ with permutation $(1, 2, 3)$, let $U = M \setminus (A_1 \cup A_2 \cup A_3)$ be the unassigned item set. The algorithm Approx3 fixes the agent order $\langle a_3, a_2, a_1 \rangle$ to apply the Round-Robin (RR) algorithm to distribute the items of U. A high-level description of the algorithm is depicted in Algorithm 3, which accepts a general well-defined allocation.

▶ **Lemma 13.** *Given a well-defined allocation $\mathcal{A} = (A_1, A_2, A_3)$ with permutation (i_1, i_2, i_3),* Approx3 *produces a complete EF1 allocation with total utility at least $\frac{1}{2} u_1(K) + \frac{1}{2} u_3(M \setminus K)$.*

Proof. Note from the RR algorithm that the returned allocation $(A_1 \cup U_1, A_2 \cup U_2, A_3 \cup U_3)$ is complete.

The initial allocation $\mathcal{A} = (A_1, A_2, A_3)$ is EF1 by Definition 12. Consider any two agents a_s and a_t, where s precedes t in the permutation (i_1, i_2, i_3). That is, a_s does not envy a_t, and thus $u_s(A_s) \geq u_s(A_t)$. Since the RR algorithm uses the agent order $\langle a_{i_3}, a_{i_2}, a_{i_1} \rangle$,

Algorithm 3 APPROX3 for a complete EF1 allocation.

Input: A well-defined allocation $\mathcal{A} = (A_1, A_2, A_3)$ with permutation (i_1, i_2, i_3).
Output: A complete EF1 allocation.

1: Fix the agent order $\langle a_{i_3}, a_{i_2}, a_{i_1} \rangle$;
2: apply the RR algorithm to allocate the unassigned items, denoted as (U_1, U_2, U_3);
3: output the final allocation $(A_1 \cup U_1, A_2 \cup U_2, A_3 \cup U_3)$.

we have $u_s(U_s) \geq u_s(U_t) - u_s(g)$, where g is the first item picked by agent a_t; and thus $u_s(A_s \cup U_s) \geq u_s(A_t \cup U_t) - u_s(g)$ for the same $g \in U_t$. That is, a_s does not strongly envy a_t. Similarly, since $\mathcal{A}$ is EF1, $u_t(A_t) \geq u_t(A_s) - u_t(g)$ for some $g \in A_s$; we have $u_t(U_t) \geq u_t(U_s)$ by the RR algorithm. Thus $u_t(A_t \cup U_t) \geq u_t(A_s \cup U_s) - u_t(g)$ for the same $g \in A_s$, that is, a_t does not strongly envy a_s. Therefore, the returned allocation is EF1.

We estimate the total utility of the returned allocation as follows. Since the agent index 1 precedes 2 in the permutation $(i_1, i_2, i_3) \in \{(1,2,3), (3,1,2)\}$, we have $u_3(U_2) \geq u_3(U_1)$ from the RR algorithm; we also have $u_1(A_1) \geq \frac{1}{2}u_1(K)$ and $u_3(A_1) \leq u_3(K)$ from Definition 12 of well-defined allocation, the total utility of the returned allocation is $u_1(A_1 \cup U_1) + u_3(A_2 \cup A_3 \cup U_2 \cup U_3)$, which is at least

$$\frac{1}{2}u_1(K) + \frac{1}{2}u_3(M \setminus A_1) = \frac{1}{2}u_1(K) + \frac{1}{2}u_3(M) - \frac{1}{2}u_3(A_1) \geq \frac{1}{2}u_1(K) + \frac{1}{2}u_3(M \setminus K).$$

This proves the lemma. ◀

▶ **Theorem 14.** *Let K be the critical set of the item g^* produced by the procedure* ALGO2. *If $\max_{g \in K} u_1(g) \geq \frac{1}{2}u_1(K)$, then* APPROX3 *returns a complete EF1 allocation with its total utility at least $\frac{1}{2}OPT$.*

Proof. For this set K, the allocation $\mathcal{A} = (\{g\}, \emptyset, \emptyset)$ is well-defined with the permutation $(1,2,3)$, where $g = \arg\max_{g \in K} u_1(g)$. The theorem follows from Lemmas 13 and 8. ◀

3.3 Case 2: $\max_{g \in K} u_1(g) < \frac{1}{2}u_1(K)$ and $u_3(g'_k) > u_3(M \setminus K)$

Recall that K is the critical set of an item g_i produced by the procedure ALGO2. In this case, $k = |K| \geq 3$ and we work with the new item order S' such that $S' \setminus g'_{k-1}$ is regular, and $g'_k \neq g_i$. Adding the last inequality $\sum_{j=1}^{k-1} u_3(g'_j) < \frac{1}{3}u_3(M \setminus g_i) + u_3(g_i)$ from Remark 11 and $u_3(g'_k) > u_3(M \setminus K)$ together gives

$$u_3(g'_k) > \frac{1}{3}u_3(M \setminus g_i) \geq u_3(K \setminus \{g'_k, g_i\}), \text{ and thus } u_3(g'_k) > \frac{1}{2}u_3(K \setminus g_i). \tag{3}$$

One sees from the above three lower bounds on $u_3(g'_k)$ that the item g'_k is very valuable to agents a_2 and a_3. We design another algorithm called APPROX7 for this case to construct a complete EF1 allocation, in which g'_k is the only item assigned to a_3.

Inside the algorithm APPROX7, the first procedure ALGO4 produces a partial EF1 and good allocation. It starts with the allocation $\mathcal{A} = (\{g'_1\}, \emptyset, \{g'_k\})$, which is EF1 and good (see Definition 2), and uses the order S' to allocate the items in the two sets $K \setminus \{g'_1 \cup g'_k\}$ and $M \setminus K$ to the agents a_1 and a_2, respectively, till exactly one of them becomes empty. A detailed description of the procedure is depicted in Algorithm 4.

▶ **Lemma 15.** *Given the critical set K for item g_i produced by the procedure* ALGO2 *satisfying $\max_{g \in K} u_1(g) < \frac{1}{2}u_1(K)$, $k = |K|$, the new item order S' in which $g'_k \neq g_i$, and $u_3(g'_k) > u_3(M \setminus K)$,* ALGO4 *produces a partial EF1 and good allocation.*

Algorithm 4 ALGO4 for a partial EF1 and good allocation.

Input: The critical set K for item g_i satisfying $\max_{g\in K} u_1(g) < \frac{1}{2}u_1(K)$, the new item order S' in which $g'_k \neq g_i$, and $u_3(g'_k) > u_3(M \setminus K)$.
Output: A partial EF1 allocation $\mathcal{A}$.

Initialize $\mathcal{A} = (\{g'_1\}, \emptyset, \{g'_k\})$ and $U = M \setminus \{g'_1, g'_k\}$;
initialize $k_1 = 2$ and $k_2 = m$;
while $(k_1 < k < k_2)$ **do**
    if (a_1 is not envied by a_2) **then**
        $A_1 = A_1 \cup \{g'_{k_1}\}$, $U = U \setminus g'_{k_1}$, and $k_1 = k_1 + 1$;
    else
        $A_2 = A_2 \cup \{g'_{k_2}\}$, $U = U \setminus g'_{k_2}$, and $k_2 = k_2 - 1$;
return $\mathcal{A}$.

Proof. Note from the description that when ALGO4 terminates, either $k_1 = k < k_2$ or $k_1 < k = k_2$, i.e., at least one item remains unassigned, and thus the returned allocation $\mathcal{A}$ is incomplete.

We remark that since the procedure starts with the allocation $(\{g'_1\}, \emptyset, \{g'_k\})$, and allocates the items in the two sets $K \setminus \{g'_1 \cup g'_k\}$ and $M \setminus K$ to the agents a_1 and a_2, respectively, the bundle $A_1 \prec A_2$ and $A_1 \prec A_3$ throughout the procedure, and thus the allocation maintains good (see Definition 2). Also throughout the procedure, since $A_3 = \{g'_k\}$ contains a single item, no one strongly envies a_3; since $u_3(g'_k) > u_3(M \setminus K)$ and Eq. (3), agent a_3 does not strongly envy any of a_1 or a_2.

We show next that throughout the procedure, between a_1 and a_2, no one strongly envies the other, and thus the final allocation is EF1. This is true for the starting allocation $(\{g'_1\}, \emptyset, \{g'_k\})$. Assume this is true for the allocation at the beginning of an iteration of the while-loop; since the allocation is good, by Lemma 3 a_1 and a_2 do not envy each other. If a_1 is not envied by a_2, then the item $g'_{k_1} \in K \setminus g'_k$ is assigned to a_1, and thus a_2 does not strongly envy a_1 due to g'_{k_1} at the end of the iteration; a_1 remains not strongly envy a_2, also due to g'_{k_1} at the end of the iteration. If a_2 is not envied by a_1, then the item $g'_{k_2} \in M \setminus K$ is assigned to a_2, and thus a_1 does not strongly envy a_2 due to g'_{k_2} at the end of the iteration; a_2 remains not strongly envy a_1, also due to g'_{k_2} at the end of the iteration. ◀

If the procedure ALGO4 terminates at $k_1 < k = k_2$, then the returned allocation is $\mathcal{A} = (\{g'_1, \ldots, g'_{k_1-1}\}, M \setminus K, \{g'_k\})$ and the unassigned item set is $U = \{g'_{k_1}, \ldots, g'_{k-1}\}$. The next procedure ALGO5 continues on to assign the items of U to produce a complete EF1 allocation, by re-setting $k_2 = k - 1$. A detailed description of ALGO5 is depicted in Algorithm 5. Basically, if at least one of a_1 and a_2 is not envied by any of the other two agents, then the procedure allocates an unassigned item to such a non-envied agent; otherwise, both a_1 and a_2 are envied by some agent and the procedure allocates all the remaining unassigned items to a_1. That is, either g'_{k_1} is assigned to a_1, or g'_{k_2} is assigned to a_2, or all of the items of $U = \{g'_{k_1}, \ldots, g'_{k_2}\}$ are assigned to a_1, respectively. One sees that the allocation $\mathcal{A}$ remains good before the last item is assigned (in fact, the allocation loses being good only if the last item is $g'_{k-1} = g_i$ and it is assigned to a_1) in the procedure ALGO5, and consequently a_1 and a_2 do not envy each other by Lemma 3 before the last item is assigned.

▶ **Lemma 16.** ALGO5 *produces a complete EF1 allocation.*

Proof. Since there will not be any unassigned item at the end of the procedure, the returned allocation is complete. We remind the readers that during the execution of ALGO5, the allocation $\mathcal{A}$ remains good before the last item is assigned.

Algorithm 5 ALGO5 for a complete EF1 allocation.

Input: The critical set K for item g_i satisfying $\max_{g\in K} u_1(g) < \frac{1}{2}u_1(K)$, the new item order S' in which $g'_{k-1} = g_i$, and $u_3(g'_k) > u_3(M \setminus K)$; the EF1 and good allocation $\mathcal{A} = (\{g'_1, \ldots, g'_{k_1-1}\}, M \setminus K, \{g'_k\})$ returned by ALGO4, where $k_1 < k$.
Output: A complete EF1 allocation $\mathcal{A}$.

Set $k_2 = k - 1$ and the unassigned item set $U = \{g'_{k_1}, \ldots, g'_{k_2}\}$;
while $(k_1 \le k_2)$ **do**
    if (a_1 is not envied) **then**
        $A_1 = A_1 \cup \{g'_{k_1}\}$, $U = U \setminus g'_{k_1}$, and $k_1 = k_1 + 1$;
    else if (a_2 is not envied) **then**
        $A_2 = A_2 \cup \{g'_{k_2}\}$, $U = U \setminus g'_{k_2}$, and $k_2 = k_2 - 1$;
    else
        $A_1 = A_1 \cup U$, and $k_1 = k_2 + 1$;
return $\mathcal{A}$.

The starting allocation $\mathcal{A} = (\{g'_1, \ldots, g'_{k_1-1}\}, M \setminus K, \{g'_k\})$, which is returned by ALGO4, is EF1 and good. At the beginning of an iteration of the while-loop, if a_1 is not envied by any other agent (the case where a_2 is not envied by any other agent is symmetric), then after g'_{k_1} is allocated to a_1 no agent will strongly envy a_1. That is, the updated allocation is EF1 too.

Consider the last case of the while-loop, i.e., at the beginning of the last iteration $\mathcal{A} = (A_1, A_2, \{g'_k\})$ is the EF1 and good allocation in which both a_1 and a_2 are envied by some agent.

We claim that a_3 envies a_2. If not, i.e., $u_3(g'_k) \ge u_3(A_2)$, then a_2 is envied by a_1, and thus by Lemma 3, a_2 does not envy a_1, i.e., $u_3(A_2) \ge u_3(A_1)$. It follows that a_3 does not envy a_1. That is, a_1 is not envied by any agent, a contradiction.

In the final allocation $\mathcal{A}' = (A_1 \cup U, A_2, \{g'_k\})$, a_1 does not strongly envy any of a_2 or a_3 for sure, and none of a_2 and a_3 strongly envies the other. From Eq. (3) a_3 does not strongly envy a_1. Since a_3 envies a_2, a_2 does not strongly envy a_1 either.

In summary, no agent strongly envies another agent in the final allocation $\mathcal{A}'$. ◀

The allocation returned by the procedure ALGO5 is complete and EF1. The next procedure ALGO6 takes in a complete EF1 allocation $\mathcal{A} = (A_1, A_2, A_3)$, and if in which agent a_1 envies a_2, i.e., $u_1(A_1) < u_1(A_2)$, then outputs the allocation between $\mathcal{A}$ and (A_2, A_1, A_3) with a larger total utility. In general, for a complete EF1 allocation $\mathcal{A} = (A_1, A_2, A_3)$, the allocation (A_2, A_1, A_3) after bundle swapping might not be EF1. We nevertheless show in the next lemma that the allocation returned by the procedure ALGO6 is EF1.

Algorithm 6 ALGO6 for a larger total utility.

Input: The complete EF1 allocation $\mathcal{A} = (A_1, A_2, A_3)$ returned by ALGO5.
Output: A complete EF1 allocation.

if $(u_1(A_1) \ge u_1(A_2))$ **then**
    output $\mathcal{A}$;
else
    output the allocation between (A_1, A_2, A_3) and (A_2, A_1, A_3) with a larger total utility.

▶ **Lemma 17.** ALGO6 *produces a complete EF1 allocation.*

Proof. Note that $\mathcal{A} = (A_2, A_1, \{g'_k\})$ can be returned only if $u_1(A_1) < u_1(A_2)$. We show that in such a case, none of a_1 and a_2 strongly envies the other in $(A_2, A_1, \{g'_k\})$ (noting that a_3 does not strongly envy a_1 or a_2, the same as in $\mathcal{A}$, and none of a_1 and a_2 strongly envies a_3 who is allocated with the single item g'_k). Because $u_1(A_2) > u_1(A_1)$, a_1 does not envy a_2 in the allocation $(A_2, A_1, \{g'_k\})$.

Let g be the last item allocated to A_2, done either by Algo4 or by Algo5. If this is done by Algo4, then (g is g'_{k+1} and) by Lines 6–7 in the description at that moment a_2 envies a_1, and thus $u_3(A_2 \setminus g) < u_3(\{g'_1, \ldots, g'_{k_1-1}\} \leq u_3(A_1)$. If g is allocated by Algo5, then by Lines 5–6 in the description at that moment a_1 is envied by some agent and a_2 is not envied by any agent, and thus by Lemma 9 $u_3(A_2 \setminus g) < u_3(\{g'_1, \ldots, g'_{k_1-1}\} \leq u_3(A_1)$. Therefore, we always have $u_3(A_1) > u_3(A_2 \setminus g)$, i.e., a_2 does not strongly envy a_1 in the allocation $(A_2, A_1, \{g'_k\})$. This proves the lemma. ◀

The algorithm, denoted as Approx7, for producing a complete EF1 allocation for Case 2, is depicted in Algorithm 7, which utilizes the three procedures introduced in the above. When Algo4 terminates at $k_1 = k < k_2$, the returned partial allocation is $\mathcal{A} = (K \setminus g'_k, A_2, \{g'_k\})$ where $A_2 \subset M \setminus K$, which is completed by calling the Envy Cycle Elimination (ECE) algorithm to assign the rest of the items, i.e., $g'_{k+1}, \ldots, g'_{k_2}$.

Algorithm 7 Approx7 for a complete EF1 allocation.

Input: The critical set K for item g_i satisfying $\max_{g \in K} u_1(g) < \frac{1}{2}u_1(K)$, the new item order S' in which $g'_k \neq g_i$, and $u_3(g'_k) > u_3(M \setminus K)$.
Output: A complete EF1 allocation.

1: Call Algo4 to produce an EF1 partial allocation $\mathcal{A} = (A_1, A_2, \{g'_k\})$, and k_1, k_2;
2: set $U = M \setminus (A_1 \cup A_2 \cup \{g'_k\})$;
3: **if** $(k_1 = k < k_2)$ **then**
4: call the ECE algorithm on $\mathcal{A}$ to continue to assign the items in U;
5: output the final allocation;
6: **else**
7: re-set $k_2 = k - 1$;
8: call Algo5 on $\mathcal{A}$ to continue to assign the items in U;
9: call Algo6 on $\mathcal{A}$ to output the final allocation.

▶ **Lemma 18.** Approx7 *produces a complete EF1 allocation.*

Proof. If the procedure Algo4 terminates at $k_1 = k < k_2$, then since the returned partial allocation is EF1 by Lemma 15, the ECE algorithm continues from there to produce a complete EF1 allocation [9].

If the procedure Algo4 terminates at $k_1 < k = k_2$, then the final allocation is returned by Algo6 and by Lemma 17 is complete and EF1. ◀

▶ **Theorem 19.** *Given the critical set K for the item g^* produced by the procedure* Algo2 *satisfying* $\max_{g \in K} u_1(g) < \frac{1}{2}u_1(K)$, $k = |K|$, *the new item order S' in which $g'_k \neq g^*$, and* $u_3(g'_k) > u_3(M \setminus K)$, Approx7 *constructs a complete EF1 allocation with its total utility at least* $\frac{1}{2}OPT$.

Proof. By Lemma 18 the final allocation is complete and EF1.

If the procedure Algo4 terminates at $k_1 = k < k_2$, then $SOL \geq u_1(K \setminus g'_k) + u_3(g'_k)$. Since $u_1(g'_k) < \frac{1}{2}u_1(K)$ and $u_3(g'_k) > u_3(M \setminus K)$, the total utility is greater than $\frac{1}{2}u_1(K) + u_3(M \setminus K) \geq \frac{1}{2}OPT$ where the last inequality holds by Lemma 8.

If the procedure ALGO4 terminates at $k_1 < k = k_2$, then let $\mathcal{A} = (A_1, A_2, \{g'_k\})$ denote the allocation returned by the procedure ALGO5, in which $A_1 \subseteq K \setminus g'_k$ and $A_2 \subset M \setminus g'_k$. We distinguish to whom g'_{k-1} is allocated.

If $g'_{k-1} \in A_1$, then $A_1 = K \setminus g'_k$, and the above argument for the first termination condition applies too, so that $SOL > \frac{1}{2}OPT$, by noting that the procedure ALGO6 never reduces the total utility.

If $g'_{k-1} \in A_2$, then $(M \setminus K) \cup \{g'_{k-1}, g'_k\} \subseteq A_2 \cup A_3$. By $\sum_{j=1}^{k-1} u_3(g'_j) < \frac{1}{3}u_3(M \setminus g^*) + u_3(g^*)$ from Remark 11, $u_3(A_2) + u_3(A_3) > \frac{2}{3}u_3(M \setminus g^*)$. When $u_1(A_1) < u_1(A_2)$, the procedure ALGO6 outputs $\mathcal{A}$ or (A_2, A_1, A_3) with a larger total utility, which is

$$SOL \geq \frac{1}{2}u_1(A_1 \cup A_2) + \frac{1}{2}u_3(A_1 \cup A_2) + u_3(A_3) \geq \frac{1}{2}u_1(M \setminus g'_k) + \frac{1}{2}u_3(M).$$

When $u_1(A_1) \geq u_1(A_2)$, $\mathcal{A}$ is the final allocation with its total utility

$$SOL \geq \frac{1}{2}u_1(A_1 \cup A_2) + u_3(A_2) + u_3(A_3) > \frac{1}{2}u_1(M \setminus g'_k) + \frac{2}{3}u_3(M \setminus g^*).$$

Noting from $g'_k \neq g^*$, $u_3(g'_k) > \frac{1}{3}u_3(M \setminus g^*)$ in Eq. (3), and $u_3(A^*_1 \setminus g^*) \leq \frac{1}{3}u_3(M \setminus g^*)$ in Lemma 6, we have $g'_k \notin A^*_1$. Therefore, $u_1(A^*_1) \leq u_1(M \setminus g'_k)$ and then since $g^* \in A^*_1$,

$$SOL \geq \frac{1}{2}u_1(M \setminus g'_k) + \min\{\frac{1}{2}u_3(M), \frac{2}{3}u_3(M \setminus g^*)\} \geq \frac{1}{2}u_1(A^*_1) + \frac{1}{2}u_3(A^*_2 \cup A^*_3) = \frac{1}{2}OPT.$$

This proves the theorem. ◀

3.4 Case 3: $\max_{g \in K} u_1(g) < \frac{1}{2}u_1(K)$ and $u_3(g'_k) \leq u_3(M \setminus K)$

Recall that K is the critical set of item g_i produced by the procedure ALGO2. The same as in Case 2, here we also have $k = |K| \geq 3$ and we work with the new item order S' such that $K = \{g'_1, \ldots, g'_k\}$, $S' \setminus g'_{k-1}$ is regular, and $g'_k \neq g_i$. Note the difference from Case 2 that, here we have $u_3(g'_k) \leq u_3(M \setminus K)$. We design an algorithm denoted APPROX9 for this case to construct a complete EF1 allocation, summarized in the following. Due to space limit, the complete design and analysis for Case 3 is in the full arXiv version [10].

▶ **Theorem 20.** *Given the critical set K for the item g^* produced by the procedure* ALGO2 *satisfying* $\max_{g \in K} u_1(g) < \frac{1}{2}u_1(K)$, *$k = |K|$, the new item order S' in which $g'_k \neq g^*$, and $u_3(g'_k) \leq u_3(M \setminus K)$,* APPROX9 *constructs a complete EF1 allocation with its total utility at least $\frac{1}{2}OPT$.*

3.5 An instance to show the tightness of ratio 2

We provide an instance below to show that the approximation ratio 2 of the combination of the three algorithms, APPROX3, APPROX7 and APPROX9, for the case of three agents with two unnormalized utility functions is tight. In this instance there are only five items in the order $\rho(g_1) > \rho(g_2) > \rho(g_3) > 1 > \rho(g_4) = \rho(g_5)$, with their values to the three agents listed as follows, where $\epsilon > 0$ is a small value:

	g_1	g_2	g_3	g_4	g_5
a_1	ϵ	1	1	0	0
a_2, a_3	0	ϵ	2ϵ	ϵ	ϵ

We continue to use the same notations introduced in Section 3. One sees that $X = \{g_1, g_2, g_3\}$, and thus $\mathcal{A}^* = (\{g_1, g_2, g_3\}, \{g_4\}, \{g_5\})$ is an optimal EF1 allocation of total utility $2 + 3\epsilon$.

For each of g_1 and g_2, $u_3(X \setminus g_i) > \frac{1}{3}u_3(M \setminus g_i) \geq \frac{4}{3}\epsilon$; the critical set produced by Algo2 is $K = \{g_1, g_2, g_3\}$. One can verify that $\max_{g \in K} u_1(g) < \frac{1}{2}u_1(K)$, and the new item order is the same as the original preference order. Since $u_3(g_3) = u_3(M \setminus K)$, it falls into Case 3. In this case, Algo8 starts with the allocation $(\{g_1\}, \{g_3\}, \emptyset)$: In the first iteration, a_1 is not envied by a_2 or a_3, the allocation is updated to $(\{g_1, g_2\}, \{g_3\}, \emptyset)$ and the procedure terminates with $k_1 > k_2$. Since $(\{g_1, g_2\}, \{g_3\}, \emptyset)$ is well-defined with permutation $(1, 2, 3)$, Approx3 calls the RR algorithm that assigns g_4 and g_5 one to each of a_3 and a_2, achieving the final allocation either $(\{g_1, g_2\}, \{g_3, g_5\}, \{g_4\})$ or $(\{g_1, g_2\}, \{g_3, g_4\}, \{g_5\})$ of total utility $1 + 5\epsilon$.

For g_3, $u_3(X \setminus g_3) = \frac{1}{3}u_3(M \setminus g_3) = \epsilon$; the critical set produced by Algo2 is also $K = \{g_1, g_2, g_3\}$, and the new item order is $S' = \langle g_1, g_3, g_2, g_4, g_5 \rangle$. Since $u_3(g_2) < u_3(M \setminus K)$, it falls into Case 3 too. In this case, Algo8 starts with the allocation $(\{g_1\}, \{g_2\}, \emptyset)$: In the first iteration, a_1 is not envied by a_2 or a_3, the allocation is updated to $(\{g_1, g_3\}, \{g_2\}, \emptyset)$ and the procedure terminates with $k_1 > k_2$. Since $(\{g_1, g_3\}, \{g_2\}, \emptyset)$ is well-defined with permutation $(1, 2, 3)$, Approx3 calls the RR algorithm that assigns g_4 and g_5 one to each of a_3 and a_2, achieving the final allocation either $(\{g_1, g_3\}, \{g_2, g_5\}, \{g_4\})$ or $(\{g_1, g_3\}, \{g_2, g_4\}, \{g_5\})$ of total utility $1 + 4\epsilon$.

For g_4 (g_5 can be identically discussed), $u_3(X \setminus g_4) = 3\epsilon > \frac{1}{3}u_3(M \setminus g_4) = \frac{4}{3}\epsilon$; the critical set produced by Algo2 is $K = \{g_1, g_2, g_3, g_4\}$, and the new item order is $S' = \langle g_1, g_2, g_4, g_3, g_5 \rangle$. Since $u_3(g_3) > u_3(M \setminus K)$, it falls into Case 2. In this case, Algo4 starts with the allocation $(\{g_1\}, \emptyset, \{g_3\})$: In the first iteration, a_1 is not envied by a_2, the allocation is updated to $(\{g_1, g_2\}, \emptyset, \{g_3\})$; in the second iteration, a_1 is envied by a_2, the allocation is updated to $(\{g_1, g_2\}, \{g_5\}, \{g_3\})$, and the procedure terminates with $k_1 < k = k_2$. Since in $(\{g_1, g_2\}, \{g_5\}, \{g_3\})$ a_1 is not envied, Algo5 assigned g_4 to a_1, achieving the complete allocation $(\{g_1, g_2, g_4\}, \{g_5\}, \{g_3\})$. Lastly, Algo6 confirms the final allocation is $(\{g_1, g_2, g_4\}, \{g_5\}, \{g_3\})$, of total utility $1 + 4\epsilon$.

Therefore, we have $SOL \leq \max\{1 + 4\epsilon, 1 + 5\epsilon\} = 1 + 5\epsilon$. It follows that $OPT/SOL \geq (2 + 3\epsilon)/(1 + 5\epsilon) = 2 - 7\epsilon/(1 + 5\epsilon) \to 2$, when ϵ tends to 0.

4 A $\frac{5}{3}$-approximation for three agents with normalized functions

Recall that the algorithm Approx1 is a 2-approximation for n agents with normalized functions. In this section, we consider the special case where $n = 3$, and again assume w.l.o.g. that agent a_1 uses the utility function $u_1(\cdot)$ and a_2 and a_3 use $u_3(\cdot)$. We continue to assume the items are given in the preference order $\rho(g_1) \geq \rho(g_2) \geq \ldots \geq \rho(g_m)$.

We also continue to use some notations and definitions introduced earlier, such as the sets $X = \{g \in M : u_1(g) \geq u_3(g)\}$ and $Y = M \setminus X$ defined in Eq. (1), a good allocation defined in Definition 2, and so on. Additionally, for any subset $A \subseteq M$, we define the quantity

$$\Delta(A) = u_1(A) - u_3(A). \tag{4}$$

Since now $u_1(M) = u_3(M) = 1$, we have $\Delta(X) + \Delta(Y) = 0$ and

$$OPT \leq u_1(X) + u_3(Y) = u_1(X) - u_3(X) + u_3(M) = \Delta(X) + 1 = 1 - \Delta(Y). \tag{5}$$

In the rest of the section we distinguish two cases on $\max_{g \in X} u_1(g)$, the most value of a single item in X to agent a_1, and we design two different, but similar, algorithms, respectively.

4.1 Case 1: $\max_{g\in X} u_1(g) \le \frac{1}{3}$

The design idea of our algorithm Approx10 is similar to Approx1, with a change that, after all the items of X have been allocated, i.e., both *to-be-allocated* items g_{k_1} and g_{k_2} are in Y, agent a_t has the priority to receive g_{k_2} if it is not envied by a_1. The detailed description of the algorithm Approx10 is presented in Algorithm 8.

Algorithm 8 Approx10 for three agents with normalized utility functions.

Input: Three agents of two types and a set of m indivisible items $\rho(g_1) \ge \rho(g_2) \ge \ldots \ge \rho(g_m)$, where $\max_{g\in X} u_1(g) \le \frac{1}{3}$.
Output: A complete EF1 allocation.

Initialize $k_1 = 1$, $k_2 = m$, and $A_i = \emptyset$ for every agent a_i;
while $(k_1 \le k_2)$ **do**
  find $t = \arg\min_{i=2,3} u_3(A_i)$;
  if $(k_1 \le |X|)$ **then** ▷ a_1 has the priority.
    if (a_1 is not envied by a_t) **then**
      $A_1 = A_1 \cup \{g_{k_1}\}$ and $k_1 = k_1 + 1$;
    else
      $A_t = A_t \cup \{g_{k_2}\}$ and $k_2 = k_2 - 1$;
  else ▷ Change of priority: Now a_t has the priority.
    if (a_t is not envied by a_1) **then**
      $A_t = A_t \cup \{g_{k_2}\}$ and $k_2 = k_2 - 1$;
    else
      $A_1 = A_1 \cup \{g_{k_1}\}$ and $k_1 = k_1 + 1$;
return the final allocation.

▶ **Lemma 21.** Approx10 *produces a good, complete and EF1 allocation.*

Proof. The returned allocation by Approx10 is clearly complete from the while-loop termination condition. We prove that at the end of each iteration of the while-loop in the algorithm, the updated allocation is good and EF1, similar to Lemma 4. Note that the initial empty allocation is trivially good and EF1.

Assume that at the beginning of the iteration the allocation denoted as $\mathcal{A} = (A_1, A_2, A_3)$ is good and EF1; note that the to-be-allocated items are g_{k_1} and g_{k_2} with $k_1 \le k_2$.

Consider the case where $k_1 > |X|$ and a_t is not envied by a_1 (we intentionally pick this case to prove, the other three cases are symmetric), in which the algorithm updates A_1 to $A_t \cup \{g_{k_2}\}$. By the description of Approx10, $A_1 = \{g_1, \ldots, g_{k_1-1}\}$ and $A_i \subseteq \{g_{k_2+1}, \ldots, g_m\}$ for $i = 2, 3$. By Definitions 1 and 2 and $k_1 \le k_2$, $A_1 \prec A_i \cup \{g_{k_2}\}$ for $i = 2, 3$, and thus the updated allocation is good.

Since at the beginning of the iteration a_t is not envied by a_1, $u_1(A_1) \ge u_3(A_t)$. Also, by the definition of t, $u_3(A_i) \ge u_3(A_t)$ for the third agent a_i. That is, no agent envies a_t in the allocation $\mathcal{A}$, and thus does not strongly envy a_t in the updated allocation (by removing the item g_{k_2}, if necessary). Note that in this iteration only a_t gets the item g_{k_2}. Therefore, the updated allocation is EF1. ◀

Let us examine one scenario of the final allocation returned by Approx10 in the next lemma, and leave the other to the main Theorem 23.

▶ **Lemma 22.** *In the final allocation $\mathcal{A} = (A_1, A_2, A_3)$ returned by* Approx10*, if $X \subseteq A_1$, then $SOL \ge \frac{2}{3}OPT$.*

Proof. If $A_1 = X$, then $\mathcal{A}$ is optimal, i.e., $SOL = OPT$.

Below we consider the scenario where a_1 receives some items from Y. This means when Approx10 terminates, $k_1 > |X|+1$ and $A_1 = \{g_1, \ldots, g_{k_1-1}\}$. Let $Y_1 = \{g_{|X|+1}, \ldots, g_{k_1-1}\}$, that is, $A_1 = X \cup Y_1$.

We claim $\min\{\Delta(X), -\Delta(Y_1)\} \leq \frac{1}{2}$. Assume to the contrary, then $u_1(X) \geq \Delta(X) > \frac{1}{2}$ and $u_3(Y_1) \geq -\Delta(Y_1) > \frac{1}{2}$. Consider the iteration where a_1 is allocated with the last item g_{k_1-1}. Since at the beginning of the iteration the allocation is good, by Lemma 3, a_1 and a_2 do not envy each other, so do a_1 and a_3. From $g_{k_1-1} \in Y$ and Lines 9–13 in the algorithm description, a_t is envied by a_1, and thus $u_1(A_1 \setminus g_{k_1-1}) < u_1(A_2 \cup A_3)$. It follows from $A_2 \cup A_3 = Y \setminus Y_1$ and Eq. (1) that

$$u_1(X) < u_1(A_2 \cup A_3) < u_3(A_2 \cup A_3) = u_3(Y) - u_3(Y_1) < 1 - \frac{1}{2} = \frac{1}{2}, \text{ a contradiction.}$$

Since $A_2 \cup A_3 \subset Y$, we have $u_1(A_2 \cup A_3) < u_3(A_2 \cup A_3)$, and therefore $SOL = u_1(A_1) + u_3(A_2 \cup A_3) > u_1(A_1) + u_1(A_2 \cup A_3) = 1$.

Using the claimed $\min\{\Delta(X), -\Delta(Y_1)\} \leq \frac{1}{2}$, if $\Delta(X) \leq \frac{1}{2}$, then by Eq. (5), $OPT \leq \frac{3}{2} \leq \frac{3}{2} \cdot SOL$; if $\Delta(X) > \frac{1}{2}$, then $-\Delta(Y_1) \leq \frac{1}{2} < \Delta(X)$, and thus $SOL = u_1(A_1) - u_3(A_1) + u_3(M) = \Delta(X) + \Delta(Y_1) + 1 > \frac{2}{3}(\Delta(X) + 1) \geq \frac{2}{3}OPT$, where the last inequality is by Eq. (5). This proves the lemma. ◀

▶ **Theorem 23.** *If* $\max_{g \in X} u_1(g) \leq \frac{1}{3}$, *then* Approx10 *produces a complete EF1 allocation with its total utility* $SOL \geq \frac{3}{5}OPT$.

Proof. Lemma 21 states that the final allocation $\mathcal{A} = (A_1, A_2, A_3)$ returned by the algorithm is complete and EF1. If $X \subseteq A_1$, then by Lemma 22 the total utility of $\mathcal{A}$ is at least $\frac{2}{3}OPT$.

We next consider the other scenario where $A_1 \subset X$, i.e., the algorithm Approx10 terminates with $k_1 \leq |X|$. Let $X_1 = \{g_{k_1}, \ldots, g_{|X|}\}$, then $A_1 = \{g_1, \ldots, g_{k_1-1}\} = X \setminus X_1$ and $A_2 \cup A_3 = X_1 \cup Y$.

We claim that $\Delta(X_1) \leq \frac{2}{3}$, and prove it by contradiction to assume $\Delta(X_1) > \frac{2}{3}$. It follows that $u_1(X_1) > \frac{2}{3}$ and thus $u_1(A_1) \leq u_1(M \setminus X_1) < \frac{1}{3}$.

From Lines 4–8 in the description of the algorithm, the item g_{k_1} is assigned to agent a_t in the last iteration of the while-loop (in which $k_2 = k_1$ at the beginning of the iteration and k_2 is decremented afterwards leading to the termination condition $k_2 < k_1$). That is, g_{k_1} is the last item received by a_t. Denote the other agent as a_i, i.e., $\{t, i\} = \{2, 3\}$. Since a_1 is envied at the beginning of the iteration, a_1 is envied by a_t; further because the allocation is good, a_1 does not envy a_t. To summarize,

$$u_3(A_t \setminus g_{k_1}) \leq u_3(A_i),\ u_3(A_t \setminus g_{k_1}) < u_3(A_1), \text{ and } u_1(A_1) \geq u_1(A_t \setminus g_{k_1}). \tag{6}$$

It follows from $A_1 \subseteq X$ that

$$u_3(A_t \setminus g_{k_1}) < u_3(A_1) \leq u_1(A_1) < \frac{1}{3}. \tag{7}$$

One sees that if agent a_i also receives some items from X_1, then the above argument applies too for the last item a_i receives, denoted as g_ℓ, such that Eq. (7) holds, i.e., $u_3(A_i \setminus g_\ell) < \frac{1}{3}$.

Subsequently, if $A_2 \cap X_1 \neq \emptyset$ and $A_3 \cap X_1 \neq \emptyset$, then

$$\Delta(X_1) \leq \Delta(X) = -\Delta(Y) \leq u_3(A_t \setminus g_{k_1}) + u_3(A_i \setminus g_\ell) < \frac{2}{3}, \text{ a contradiction.}$$

If $X_1 \subseteq A_t$, then by Eq. (6),

$$u_1(g_{k_1}) \geq u_1(A_t) - u_1(A_1) \geq u_1(X_1) - u_1(A_1) > \frac{1}{3}, \text{ contradicting to } \max_{g \in X} u_1(g) \leq \frac{1}{3}.$$

We have thus proved the claim that $\Delta(X_1) \leq \frac{2}{3}$.

We linearly combine $\Delta(X_1) \leq \frac{2}{3}$ and $\Delta(X_1) \leq \Delta(X)$ to have $\Delta(X_1) \leq \frac{3}{5} \cdot \frac{2}{3} + \frac{2}{5} \cdot \Delta(X) = \frac{2}{5} + \frac{2}{5}\Delta(X)$, and thus by Eq. (5)

$$SOL = u_1(X \setminus X_1) + u_3(X_1 \cup Y) = 1 + \Delta(X) - \Delta(X_1) \geq \frac{3}{5}(1 + \Delta(X)) \geq \frac{3}{5}OPT.$$

This proves the theorem. ◀

4.2 Case 2: $\max_{g \in X} u_1(g) > \frac{1}{3}$

In this case, we let $g^* = \arg\max_{g \in X} u_1(g)$; thus $u_1(g^*) > \frac{1}{3}$, which is so big that it is assigned to agent a_1 immediately. On the other hand, we can use g^* to bound OPT as follows:

$$OPT \leq u_1(X) + u_3(Y) \leq u_1(X) + u_3(M) - u_3(g^*) \leq 2 - u_3(g^*) < \frac{5}{3} + \Delta(g^*). \tag{8}$$

Our algorithm for this case is very similar to Approx10, with two changes: One is the starting allocation set to be $(\{g^*\}, \emptyset, \emptyset)$, and the other is, when $k_2 = |X|$, the while-loop terminates and the algorithm switches to the Envy-Cycle Elimination (ECE) algorithm to assign the rest of items. The detailed description of the algorithm, denoted as Approx11, is presented in Algorithm 9.

Algorithm 9 Approx11 for three agents with normalized utility functions.

Input: Three agents of two types and a set of m indivisible items $\rho(g_1) \geq \rho(g_2) \geq \ldots \geq \rho(g_m)$, where $g^* = \arg\max_{g \in X} u_1(g)$ and $u_1(g^*) > \frac{1}{3}$.
Output: A complete EF1 allocation.

Initialize $k_1 = 1$, $k_2 = m$, and $\mathcal{A} = (\{g^*\}, \emptyset, \emptyset)$;
if $g_{k_1} = g^*$, then $k_1 = k_1 + 1$;
while ($k_1 \leq k_2$ and $k_2 > |X|$) **do**
    find $t = \arg\min_{i=2,3} u_3(A_i)$;
    if ($k_1 \leq |X|$) **then**
        if (a_1 is not envied by a_t) **then**
            $A_1 = A_1 \cup \{g_{k_1}\}$, $k_1 = k_1 + 1$, if $g_{k_1} = g^*$ then $k_1 = k_1 + 1$;
        else
            $A_t = A_t \cup \{g_{k_2}\}$ and $k_2 = k_2 - 1$;
    else
        if (a_t is not envied by a_1) **then**
            $A_t = A_t \cup \{g_{k_2}\}$ and $k_2 = k_2 - 1$;
        else
            $A_1 = A_1 \cup \{g_{k_1}\}$, $k_1 = k_1 + 1$, if $g_{k_1} = g^*$ then $k_1 = k_1 + 1$;
if ($k_2 = |X|$) **then**
    call the ECE algorithm on $\mathcal{A}$ to continue to assign the items in $\{g_{k_1}, \ldots, g_{k_2}\} \setminus \{g^*\}$;
return the final allocation.

▶ **Theorem 24.** *If $\max_{g \in X} u_1(g) > \frac{1}{3}$, then* Approx11 *produces a complete EF1 allocation with its total utility $SOL \geq \frac{3}{5}OPT$, and the approximation ratio $\frac{3}{5}$ is tight.*

Proof. We distinguish the two termination condition of the while-loop in Approx11.

In the first scenario, the while-loop terminates at $k_1 > k_2$, that is, all the items are assigned and $k_2 \geq |X|$ implying $X \subseteq A_1$ in the returned allocation $\mathcal{A} = (A_1, A_2, A_3)$. Note that the while-loop body is the same as the while-loop body in Approx10. It follows from Lemma 21 that $\mathcal{A}$ is good, complete and EF1, and from Lemma 22 that its total utility is $SOL \geq \frac{2}{3}OPT$.

In the other scenario, the while-loop terminates at $k_1 \leq k_2 = |X|$, that is, all the items of Y have been assigned to a_2 and a_3 (i.e., $Y \subseteq A_2 \cup A_3$), and the unassigned items are $g_{k_1}, \ldots, g_{k_2}$ excluding g^*. By Lemma 21, the achieved allocation $\mathcal{A}$ is good and EF1. Since $g^* \in A_1$, we have $\Delta(A_1) \geq \Delta(g^*)$.

We claim that the ECE algorithm does not decrease $\Delta(A_1)$. To prove the claim, we assume in an iteration of the ECE algorithm, with the allocation $\mathcal{A} = (A_1, A_2, A_3)$ at the beginning, agent a_1 gets the bundle A_2 or A_3 due to the existence of an envy cycle. Further assume w.l.o.g. that a_1 gets bundle A_2, which means a_1 envies a_2 and the envy cycle is either $(1 \to 2 \to 3 \to 1)$ or $(1 \to 2 \to 1)$. Either way, $u_1(A_1) < u_1(A_2)$ and $u_3(A_2)(< u_3(A_3)) < u_3(A_1)$. Therefore, $\Delta(A_1) < \Delta(A_2)$. This proves the claim.

The final allocation $\mathcal{A} = (A_1, A_2, A_3)$ returned from the ECE algorithm has its total utility

$$SOL = u_1(A_1) + u_3(A_2 \cup A_3) \geq \Delta(A_1) + u_3(M) \geq \Delta(g^*) + 1 > \frac{3}{5}OPT,$$

where the last inequality is by Eq. (8). We thus prove that the approximation ratio of Approx11 is at least $\frac{3}{5}$.

We next provide an instance below to show that this approximation ratio is tight. In this instance, there are five items given in the order $\rho(g_1) > \rho(g_2) > \rho(g_3) > 1 > \rho(g_4) = \rho(g_5)$, with their values to the three agents listed as follows, where $\epsilon > 0$ is a small value:

	g_1	g_2	g_3	g_4	g_5
a_1	$\frac{1}{3}$	$\frac{1}{3} - \epsilon$	$\frac{1}{3} + \epsilon$	0	0
a_2, a_3	ϵ	ϵ	$\frac{1}{3}$	$\frac{1}{3} - \epsilon$	$\frac{1}{3} - \epsilon$

One sees that the instance is normalized, $X = \{g_1, g_2, g_3\}$, and $A^* = (\{g_1, g_2\}, \{g_3, g_4\}, \{g_5\})$ is an optimal EF1 allocation of total utility $\frac{5}{3} - 3\epsilon$.

Since $u_1(g_3) > \frac{1}{3}$, Approx11 is executed which starts with the allocation $(\{g_3\}, \emptyset, \emptyset)$: In the first iteration, a_1 is envied by a_2 and a_3, and the allocation is updated to $(\{g_3\}, \emptyset, \{g_5\})$ or $(\{g_3\}, \{g_5\}, \emptyset)$; in the second iteration, a_1 and a_3 (resp., a_1 and a_2) are envied by a_2 (resp., a_3), and the allocation is updated to $\mathcal{A} = (\{g_3\}, \{g_4\}, \{g_5\})$; the procedure terminates with $k_2 = |X| = 3$.

Next, the ECE algorithm is called on $\mathcal{A}$ to continue to assign the items g_1 and g_2: In the first iteration, since in $(\{g_3\}, \{g_4\}, \{g_5\})$ only a_1 is still envied by a_2 and a_3, a_2 and a_3 are not envied, the ECE algorithm assigns g_1 to a_2 (or to a_3, symmetrically) resulting in the updated allocation $(\{g_3\}, \{g_1, g_4\}, \{g_5\})$; in the second iteration, the ECE algorithm assigns the last item g_2 to a_3, ending at the complete allocation $\mathcal{A} = (\{g_3\}, \{g_1, g_4\}, \{g_2, g_5\})$. This final allocation $\mathcal{A} = (\{g_3\}, \{g_1, g_4\}, \{g_2, g_5\})$ has total utility 1. Therefore, we have $OPT/SOL = (\frac{5}{3} - 3\epsilon)/1 \to \frac{5}{3}$, when ϵ tends to 0. ◀

5 Conclusion

Fair division of indivisible goods is an interesting problem which has received a lot of studies, from multiple research communities including artificial intelligence and theoretical computer science. Finding an EF1 allocation to maximize the utilitarian social welfare from the

perspective of approximation algorithms emerges most recently. Despite EF1 being one of the most simplest fairness criteria, the design and analysis of approximation algorithms for this problem in the general case seems challenging [4, 6]. In this paper, we focused on the special case where agents are of only two types, and we hope our work may shed lights on and inspire more studies for the general case. For this special case, we presented a 2-approximation algorithm for any number of agents with normalized utility functions; by the lower bound of $\frac{4n}{3n+1}$ from [6], our result shows the problem in this special case is APX-complete. When there are only three agents, we presented an improved $\frac{5}{3}$-approximation algorithm. When there are only three agents but the utility functions are unnormalized, we presented a tight 2-approximation algorithm which is the best possible by the lower bound of $\frac{1+\sqrt{4n-3}}{2}$ from [6]. In all three algorithms, we demonstrate the use of the item *preference* (Definition 1) order, which can be explored further for improved algorithms.

We remark that the lower bound of $\frac{4n}{3n+1}$ on the approximability in [6] is proven for a more restricted case where the two utility functions are normalized, agent a_1 uses a utility function and all the other agents use the other function. It would be interesting to narrow the gap for either the special case we study or this more restricted case; we expect some improved lower bounds or some approximation ratios as functions in n_1 and n_2, where n_1 agents use a common utility function and the other $n_2 = n - n_1$ use the second utility function.

References

1 G. Amanatidis, H. Aziz, G. Birmpas, A. Filos-Ratsikas, B. Li, H. Moulin, A. A. Voudouris, and X. Wu. Fair division of indivisible goods: Recent progress and open questions. *Artificial Intelligence*, 322:Article 103965, 2023.

2 G. Amanatidis, G. Birmpas, A. Filos-Ratsikas, and A. Voudouris. Fair division of indivisible goods: A survey. In *Proceedings of IJCAI 2022*, pages 5385–5393, 2022.

3 H. Aziz, X. Huang, N. Mattei, and E. Segal-Halevi. Computing welfare-maximizing fair allocations of indivisible goods. *European Journal of Operational Research*, 307:773–784, 2023. doi:10.1016/J.EJOR.2022.10.013.

4 S. Barman, U. Bhaskar, and N. Shah. Optimal bounds on the price of fairness for indivisible goods. In *Proceedings of WINE 2020*, pages 356–369, 2020.

5 X. Bei, X. Lu, P. Manurangesi, and W. Suksompong. The price of fairness for indivisible goods. *Theory of Computing Systems*, 65:1069–1093, 2021. doi:10.1007/S00224-021-10039-8.

6 X. Bu, Z. Li, S. Liu, J. Song, and B. Tao. Approximability landscape of welfare maximization within fair allocations. In *Proceedings of EC 2025*, pages 412–440, 2025.

7 E. Budish. The combinatorial assignment problem: Approximate competitive equilibrium from equal incomes. *Journal of Political Economy*, 119:1061–1103, 2011.

8 I. Caragiannis, D. Kurokawa, H. Moulin, A. Procaccia, N. Shah, and J. Wang. The unreasonable fairness of maximum Nash welfare. *ACM Transactions on Economics and Computation*, 7:12:1–12:32, 2019. doi:10.1145/3355902.

9 R. Lipton, E. Markakis, E. Mossel, and A. Saberi. On approximately fair allocations of indivisible goods. In *Proceedings of EC 2004*, pages 125–131, 2004.

10 J. Ma, Y. Chen, G. Chen, M. Gong, G. Lin, and A. Zhang. Maximizing social welfare among EF1 allocations at the presence of two types of agents. *CoRR*, 2025. arXiv:2509.09641.

11 H. Steinhaus. Sur la division pragmatique. *Econometrica*, 17:315–319, 1949.

12 H. Varian. Equity, envy and efficiency. *Journal of Economic Theory*, 9:63–91, 1974.

Matchgate Signatures Under Variable Permutations

Boning Meng[1]
Key Laboratory of System Software (Chinese Academy of Sciences) and State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
University of Chinese Academy of Sciences, Beijing, China

Yicheng Pan[2]
State Key Laboratory of Complex & Critical Software Environment, Beihang University, Beijing, China

Abstract

In this work, we introduce the concept of permutable matchgate signatures and leverage it to establish dichotomy theorems for #CSP and $\#R_D$-CSP ($D \geq 3$) on planar graphs without the variable ordering restriction. We also present a complete characterization of permutable matchgate signatures and their relationship to symmetric signatures. Besides, we give a sufficient and necessary condition for determining whether a matchgate signature retains its property under a certain variable permutation, which can be checked in polynomial time. In addition, we prove a dichotomy for Pl-$\#R_D$-CSP ($D \geq 3$), where the variable ordering restriction exists.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases Computational Complexity, Matchgate Signature, Counting CSP

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.50

Related Version *Full Version*: `https://arxiv.org/abs/2503.21194`

Funding *Boning Meng*: supported by National Key R&D Program of China (2023YFA1009500), NSFC 62532014 and NSFC 62272448.
Yicheng Pan: supported by State Key Laboratory of Complex & Critical Software Environment (SKLCCSE-2024ZX-20) and NSFC 61932002.

Acknowledgements We want to thank Mingji Xia for introducing this problem to us.

1 Introduction

Counting perfect matchings in a graph (denoted by #PM) is of great significance in counting complexity. The study of #PM was motivated by the dimer problem in statistical physics [13, 14, 15, 19], and two fundamental results emerged from this study. The first breakthrough occurred in 1961, when a polynomial time algorithm for #PM on planar graphs was developed by Kasteleyn, Temperley and Fisher [14, 19], now known as the FKT algorithm. The second significant advancement occurred in 1979, when Valiant defined the complexity class #P and proved that #PM on general graphs is #P-hard [21]. #PM was the first natural counting problem discovered to be #P-hard on general graphs and polynomial-time computable on planar graphs.

Matchgates were later introduced to generalize the FKT algorithm. Multiple studies have been undertaken to systematically define and characterize matchgates [22, 23, 24, 2, 4, 18, 7, 16]. In particular, the matchgate is proved to be highly related to the complexity classification for counting constraint satisfaction problems (denoted by #CSP) on plane graphs over Boolean domain and complex range [12, 5].

[1] First author.
[2] Corresponding author.

© Boning Meng and Yicheng Pan;
licensed under Creative Commons License CC-BY 4.0

In this article, we always restrict ourselves to Boolean domain and complex range. We further develop the theory of matchgates and establish new complexity classifications for a variant of #CSP. Our primary contribution is the introduction and detailed characterization of permutable matchgate signatures – a novel concept that enables complexity analyses for #CSP variants, such as #CSP on planar graphs without the variable ordering restriction.

Section 1.1 and Section 1.2 introduce the existing results regarding matchgates and #CSP. Section 1.3 explains our motivation and presents our results.

1.1 #PM and matchgate signatures

For a graph $G = (V, E)$, a *matching* is an edge set $M \subseteq E$ such that no pair of edges in M shares a common endpoint. Besides, if the vertices that M contains are exactly $V(M) = V$, then M is a *perfect matching* of G.

▶ **Definition 1.** *An instance of #PM is a graph $G = (V, E)$ with weighted edges $w : E \to \mathbb{C}$. The weight of a matching M is $w(M) = \prod_{e \in M} w(e)$. The output of the instance is the sum of the weights of all perfect matchings in G:*

$$\#PM(G) = \sum_{M:M \text{ is a perfect matching of } G} w(M).$$

When $w(e) = 1$ for each $e \in E$, the output of the instance is exactly the number of perfect matchings in the graph.

Matchgate and their associated matchgate signatures are defined in the context of #PM. A *signature* (also referred to as a *constraint function* in some works) is defined as a function $f : \{0, 1\}^k \to \mathbb{C}$ that maps a string of length k to a complex number. For $G = (V, E)$ and $X \subseteq V$, we use $G - X$ to denote the graph obtained by deleting vertices in X from G.

▶ **Definition 2.** *A matchgate is a plane graph $G = (V, E)$ with weighted edges $w : E \to \mathbb{C}$, and together with some external nodes $U \subseteq V$ on its outer face labelled by $\{1, 2, ..., |U|\}$ in a clockwise order. The signature f of a matchgate G is a Boolean signature of arity $|U|$ and for each $\alpha \in \{0, 1\}^{|U|}$,*

$$f(\alpha) = \#PM(G - X),$$

where $X \subseteq U$ and a vertex in U with label i belongs to X if and only if the ith bit of α is 1.

A signature f is a matchgate signature if it is the signature of some matchgate. $\mathscr{M}$ denotes all the matchgate signatures.

Matchgates provide a generalization of the FKT algorithm in the following way. Suppose G is a plane graph with each vertex v representing some matchgate signature, forming a signature grid. By replacing each vertex v with its corresponding matchgate H, the resulting graph G' remains a plane graph and consequently $\#PM(G')$ can be solved in polynomial time by the FKT algorithm. Consequently, the value of the signature grid G, which is exactly $\#PM(G')$, can be computed in polynomial time.

In particular, a so-called *matchgate identity* (MGI) has been verified to be the necessary and sufficient condition for a signature to be a matchgate signature [2, 4, 7], which provides an algebraic way to characterize matchgate signatures. This identity also enables an universal way to construct a corresponding matchgate for a matchgate signature.

▶ **Theorem 3** (MGI)**.** *Suppose f is a signature of arity k. Then f is a matchgate signature if and only if the following identity, denoted by MGI, is satisfied:*

For each $\beta, \gamma \in \{0,1\}^k$, let $P = \{1 \le j \le k | \beta_j \neq \gamma_j\}$, $l = |P|$. Let $p_j \in P$ be the jth smallest number in P and let $e_{p_j} \in \{0,1\}^k$ denotes a string with a 1 in the p_jth index, and 0 elsewhere. Then

$$\sum_{j=1}^{n} (-1)^j f(\beta \oplus e_{p_j}) f(\gamma \oplus e_{p_j}) = 0.$$

▶ **Lemma 4** ([7]). *A matchgate signature of arity k can be realized by a matchgate with at most $O(k^4)$ vertices, which can be constructed in $O(k^4)$ time.*

1.2 #CSP

A *counting constraint satisfaction problem* is specified by a signature set $\mathcal{F}$. #CSP($\mathcal{F}$) asks for the value of an instance, which is the sum of the values over all configurations. Here, $\mathcal{F}$ is a fixed and finite set of signatures. An instance of #CSP($\mathcal{F}$) is specified as follows:

▶ **Definition 5** (#CSP [10]). *An instance I of #CSP($\mathcal{F}$) has n variables and m signatures from $\mathcal{F}$ depending on these variables. The value of the instance then can be written as*

$$Z(I) = \sum_{(x_1,\ldots,x_n)\in\{0,1\}^n} \prod_{1\le i\le m} f_i(x_{i_1}, \ldots, x_{i_k}),$$

where $f_1, \ldots, f_m$ are signatures in I and f_i depends on $x_{i_1}, \ldots, x_{i_k}$ for each $1 \le i \le m$.

The *underlying graph* of a #CSP($\mathcal{F}$) instance I is its incidence graph. It is a bipartite graph $G = (U, V, E)$, where for every constraint f there is a $u_f \in U$, for every variable x there is a $v_x \in V$, and $(u_f, v_x) \in E$ if and only if f depends on x. See Figure 1 for an example. Sometimes we also denote the value $Z(I)$ as $Z(G)$ for convenience.

There are several important variants of #CSP. A signature f is said to be *symmetric* if the output of f depends only on the Hamming weight of the input. If each signature in $\mathcal{F}$ is restricted to be symmetric, this kind of problem is denoted by symmetric #CSP, or sym-#CSP for short. If the maximum degree of the vertices in V is at most a constant $D \ge 3$, this kind of problem is denoted by #R_D-CSP [9]. If each vertex in V is of degree 2, this kind of problem is denoted by Holant [8] (See Definition 15 for details). If we restrict G to be a plane graph, in which the variables that f_u depends on are ordered clockwise for each $u \in U$ (denoted by *the variable ordering restriction*), then we denote this problem as Pl-#CSP. Pl-#R_D-CSP is defined similarly.

To study the complexity of these problems, a number of dichotomy theorems have been established, which classify that for each signature set, the specified problem is either polynomial-time computable or #P-hard. $\mathscr{A}$ and $\mathscr{P}$ are two fundamental tractable signature sets, whose definition can be found in [9]. $\widehat{\mathscr{M}}$ denote the matchgate signatures under a specific holographic transformation explained later in Section 2.2.2.

▶ **Theorem 6** ([9]). *If $\mathcal{F} \subseteq \mathscr{A}$ or $\mathcal{F} \subseteq \mathscr{P}$, then #CSP($\mathcal{F}$) is polynomial-time computable; otherwise it is #P-hard.*

▶ **Theorem 7** ([9]). *Suppose $D \ge 3$ is an integer. If $\mathcal{F} \subseteq \mathscr{A}$ or $\mathcal{F} \subseteq \mathscr{P}$, then #$R_D$-CSP($\mathcal{F}$) is polynomial-time computable; otherwise it is #P-hard.*

▶ **Theorem 8** ([5]). *If $\mathcal{F} \subseteq \mathscr{A}$ or $\mathcal{F} \subseteq \mathscr{P}$ or $\mathcal{F} \subseteq \widehat{\mathscr{M}}$, then Pl-#CSP($\mathcal{F}$) is polynomial-time computable; otherwise it is #P-hard.*

1.3 Motivation and results

The study of counting complexity has been extended to a number of different graph classes. In particular, the complexity of #PM on minor-excluded graphs has been fully classified by [11, 20]. These results generalize the FKT algorithm and the #P-hardness of #PM on general graphs. It is natural to consider adapting other counting algorithms and hardness results related to #PM, such as the dichotomy for #CSP, to the minor-excluded setting. However, existing results on Pl-#CSP rely on the variable ordering restriction, a restriction for the signatures rather than the graph class. This limitation obstructs the direct extension of Pl-#CSP to broader graph families.

To overcome this limitation, we focus on #CSP on planar graphs without the variable ordering restriction. We use $\mathcal{PL}$ to denote the class of planar graphs and $\#\mathrm{CSP}(\mathcal{F})\langle\mathcal{PL}\rangle$ to denote #CSP specified by the signature set $\mathcal{F}$ over the graph class $\mathcal{PL}$. $\#\mathrm{CSP}(\mathcal{F})\langle\mathcal{PL}\rangle$ is exactly #CSP on planar graphs without the variable ordering restriction.

Before investigating $\#\mathrm{CSP}\langle\mathcal{PL}\rangle$, we also observe that the complexity classification for Pl-$\#R_D$-CSP remains open in the literature. To address this gap, we establish the following result:

▶ **Theorem 9.** *Suppose $D \geq 3$ is an integer. If $\mathcal{F} \subseteq \mathscr{A}$ or $\mathcal{F} \subseteq \mathscr{P}$ or $\mathcal{F} \subseteq \widehat{\mathscr{M}}$, Pl-$\#R_D$-CSP($\mathcal{F}$) is polynomial-time computable; otherwise it is #P-hard.*

The full proof of Theorem 9 is similar to that of Theorem 7 originally presented in in [9, Section 6], and can be found in the full version. Here, we only remark the major differences between the two proofs.

- All the gadgets in our proof are constructed in the setting of Pl-$\#R_D$-CSP instead of $\#R_D$-CSP;
- Unlike [9, Lemma 6.5], we realize $[0, 0, 1] = [0, 1]^{\otimes 2}$ instead of $[0, 1]^{\otimes m}$;
- Unlike [9, Lemma 6.2], instead of realizing a single non-degenerate binary signature h, we realize $h^{\otimes 2}$ and use them to form a non-degenerate binary signature $g = h^T h$.

Returning to the framework of $\#\mathrm{CSP}\langle\mathcal{PL}\rangle$, we demonstrate that the concept of permutable matchgate signatures plays a pivotal role in establishing its dichotomy theorem in a straightforward way.

▶ **Definition 10** (Permutable matchgate signature)**.** *Suppose f is a signature of arity n. For a permutation $\pi \in S_n$, we use f_π to denote the signature*

$$f_\pi(x_1, \ldots, x_n) = f(\pi(x_1), \ldots, \pi(x_n))$$

If for each $\pi \in S_n$, f_π is a matchgate signature, we say f is a permutable matchgate signature.

We use $\mathscr{M}_P$ to denote the set of all the permutable matchgate signatures.

▶ **Theorem 11.** *Let $\mathcal{F}$ be a finite signature set and $D \geq 3$ be an integer.*

If $\mathcal{F} \subseteq \mathscr{A}$ or $\mathcal{F} \subseteq \mathscr{P}$ or $\mathcal{F} \subseteq \widehat{\mathscr{M}_P}$, $\#CSP(\mathcal{F})\langle\mathcal{PL}\rangle$ is polynomial-time computable; otherwise it is #P-hard.

If $\mathcal{F} \subseteq \mathscr{A}$ or $\mathcal{F} \subseteq \mathscr{P}$ or $\mathcal{F} \subseteq \widehat{\mathscr{M}_P}$, $\#R_D\text{-}CSP(\mathcal{F})\langle\mathcal{PL}\rangle$ is polynomial-time computable; otherwise it is #P-hard.

Proof. Let $\mathcal{F}' = \{f_\pi | f \in \mathcal{F}, \mathrm{arity}(f) = n, \pi \in S_n\}$. Then we have $\#\mathrm{CSP}(\mathcal{F})\langle\mathcal{PL}\rangle \equiv_T$ Pl-$\#\mathrm{CSP}(\mathcal{F}')$. By Definitions of $\mathscr{A}$ and $\mathscr{P}$, $\mathcal{F}' \subseteq \mathscr{A}$ or $\mathscr{P}$ if and only if $\mathcal{F} \subseteq \mathscr{A}$ or $\mathscr{P}$ respectively. Furthermore, $\mathcal{F}' \subseteq \widehat{\mathscr{M}}$ if and only if $\mathcal{F} \subseteq \widehat{\mathscr{M}_P}$ by Definition 10. Consequently, we are done by replacing the tractable criteria for $\mathcal{F}'$ in Theorem 8 with those for $\mathcal{F}$. The same argument holds for $\#R_D\text{-}\mathrm{CSP}\langle\mathcal{PL}\rangle$ as well. ◀

The main contribution of this article is a detailed characterization of permutable matchgate signatures, showing the connection between them and symmetric matchgate signatures. This characterization has been proved to be useful in the algorithm design and the hardness proof in the subsequent work [17]. A signature f is said to be realized by $\mathcal{F}$ if it can be simulated using signatures in $\mathcal{F}$ under polynomial-time Turing reduction.

▶ **Theorem 12** (Informal version of Theorem 25). *Each permutable matchgate signature of arity n can be realized by a symmetric matchgate signature of arity n and $O(n)$ binary matchgate signatures.*

▶ **Theorem 13** (Informal version of Theorem 26). *For each permutable matchgate signature f, if $\#R_3$-CSP$(\{f\})$ is #P-hard, then a symmetric matchgate signature g, satisfying $\#R_3$-CSP$(\{g\})$ is #P-hard, can be realized by f and 3 specific matchgate signatures $[1,0],[1,0,1],[1,0,1,0]$.*

It is noteworthy that there are actually three types of permutable matchgate signatures (Pinning, Parity, Matching) possess different properties, but they are all related to the corresponding symmetric matchgate signatures by our proof.

In addition, we also present a sufficient and necessary condition for determining whether a matchgate signature retains its property under a specific variable permutation. This condition is a simplified version of the MGI property, since it only requires considering $O(n^4)$ equations from MGI rather than all $O(2^n)$ equations.

▶ **Theorem 14** (Informal version of Theorem 22). *Suppose f is a matchgate signature satisfying $f(\vec{0}) \neq 0$ and π is a permutation. Then f_π retains MGI if $O(n^4)$ specific equations from MGI are still satisfied.*

The proof of this theorem is inspired by an alternative proof of Theorem 3, given by Jerrum and recorded in [1, Section 4.3.1]. By dividing the summation in MGI into appropriate parts, we prove this result by a non-trivial induction.

2 Preliminaries

2.1 Counting problems

For a string $\alpha = \alpha_1 \dots \alpha_k \in \{0,1\}^k$, the *Hamming weight* of α is the number of 1s in α, denoted by $HW(\alpha)$. We use $\overline{\alpha}$ to denote the string that differs from α at every bit, which means $\alpha_i + \overline{\alpha}_i = 1$ for each $1 \leq i \leq k$.

For a signature $f : \{0,1\}^k \to \mathbb{C}$, k is denoted by the *arity* of f. A symmetric signature f of arity k can be denoted by $[f_0, f_1, ..., f_k]_k$, or simply $[f_0, f_1, ..., f_k]$ when k is clear from the context, where for $0 \leq i \leq k$, f_i is the value of f when the Hamming weight of the input is i. For $c \in \mathbb{C}$, we also use the notation $c[f_0, f_1, ..., f_k]$ to denote the signature $[cf_0, cf_1, ..., cf_k]$. We use $\leq_T$ and $\equiv_T$ to respectively denote polynomial-time Turing reduction and equivalence. We denote by $f^{x_i=c}$ the signature that pins the ith variable to $c \in \{0,1\}$:

$$f^{x_i=c}(x_1, ..., x_{i-1}, x_{i+1}, ..., x_k) = f(x_1, ..., x_{i-1}, c, x_{i+1}, ..., x_k).$$

2.1.1 Holant problems

A *Holant problem* Holant$(\mathcal{F})$ can be seen as a #CSP$(\mathcal{F})$ problem with the restriction that all the variables must appear exactly twice.

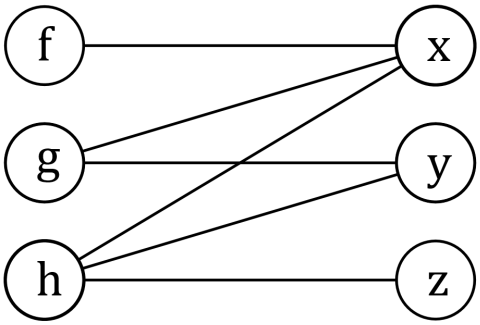


Figure 1 The underlying graph of a #CSP($\mathcal{F}$) instance if $f, g, h \in \mathcal{F}$ and x, y, z are 3 variables; or that of a Holant($\mathcal{F}|\mathcal{EQ}$) instance if we treat x, y, z as $=_3, =_2, =_1$ and the edges as the variables instead.

▶ **Definition 15.** *An instance of Holant($\mathcal{F}$) has an underlying graph $G = (V, E)$. Each vertex $v \in V$ is assigned a signature from $\mathcal{F}$ and each edge in E represents a variable. Here, $\mathcal{F}$ is a fixed set of signatures and usually finite. The signature assigned to the vertex v is denoted by f_v. An assignment of E is a mapping $\sigma : E \to \{0, 1\}$, which can also be expressed as an assignment string $\sigma \in \{0, 1\}^{|E|}$, and the value of the assignment is defined as*

$$\omega(\sigma) = \prod_{v \in V} f_v(\sigma),$$

where $f_v(\sigma) = f_v(\sigma(e_{v_1}), ..., \sigma(e_{v_k}))$ and v is incident to $e_{v_1}, ..., e_{v_k}$.

The output of the instance, or the value of G, is the sum of the values of all possible assignments of E, denoted by:

$$Z(G) = \sum_{\sigma \in \{0,1\}^{|E|}} \omega(\sigma).$$

Furthermore, we use Holant($\mathcal{F}_1|\mathcal{F}_2$) represents Holant($\mathcal{F}_1 \cup \mathcal{F}_2$) with the restriction that the underlying graph $G = (U, V, E)$ is bipartite, and each vertex $u \in U$ is assigned a signature from $\mathcal{F}_1$ while each vertex $v \in V$ is assigned a signature from $\mathcal{F}_2$. We denote by $\mathcal{EQ}$ the set of all equality functions. In other words, $\mathcal{EQ} = \{=_k | k \geq 1\}$ where $=_k$ is the signature $[1, 0, ..., 0, 1]_k$. We also denote $\{=_k | 1 \leq k \leq D\}$ by $\mathcal{EQ}_{\leq D}$ for each integer $D \geq 1$. By definition, we have the following lemma. Also see Figure 1 for an example.

▶ **Lemma 16.** *Let $\mathcal{C}$ be an arbitrary graph class, $\mathcal{F}$ be an arbitrary signature set and $D \geq 1$ be an integer. Then,*

$$\#CSP(\mathcal{F})\langle\mathcal{C}\rangle \equiv_T Holant(\mathcal{F}|\mathcal{EQ})\langle\mathcal{C}\rangle,$$

$$\#R_D\text{-}CSP(\mathcal{F})\langle\mathcal{C}\rangle \equiv_T Holant(\mathcal{F}|\mathcal{EQ}_{\leq D})\langle\mathcal{C}\rangle.$$

2.2 Reduction methods

2.2.1 Constructing gadgets

A *gadget* of Holant($\mathcal{F}$) has an underlying graph $GG = (V, E, D)$, where E is the set of normal edges and D is the set of edges with only one endpoint, called *dangling edges* [3]. Each vertex in GG is still assigned a signature from $\mathcal{F}$. A signature f of arity $|D|$ is said to be *realized*

[3] In order to differentiate from the notation of a graph, we use two capital letters to represent a gadget.

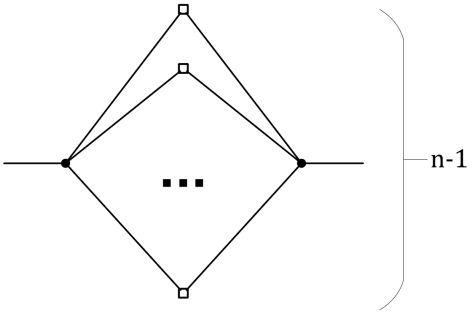


Figure 2 The construction of the generalized mating gadget.

by $GG = (V, E, D)$, if for each assignment $\alpha : D \to \{0, 1\}$, $f(\alpha) = \sum_{\sigma \in \{0,1\}^{|E|}} \omega(\alpha\sigma)$, where $\alpha\sigma$ is the assignment of edges in $D \cup E$. In this case, we also say f can be realized by $\mathcal{F}$. By constructing gadgets with existing signatures, we are able to realize desired signatures.

▶ **Lemma 17.** *If f can be realized by $\mathcal{F}$, then Holant$(\mathcal{F}) \equiv_T$ Holant$(\mathcal{F} \cup \{f\})$.*

Also, we present some derivative concepts related to the concept of a gadget. A *left-side gadget* of Holant$(\mathcal{F}_1|\mathcal{F}_2)$ has a bipartite underlying graph $GG = (U, V, E, D)$, where each vertex $u \in U$ is assigned a signature from $\mathcal{F}_1$, each vertex $v \in V$ is assigned a signature from $\mathcal{F}_2$, E is the set of normal edges and D is the set of dangling edges. Furthermore, the endpoint of each dangling edge must belong to U. It is easy to verify that, if f can be realized by GG, then Holant$(\mathcal{F}_1|\mathcal{F}_2) \equiv_T$ Holant$(\mathcal{F}_1 \cup \{f\}|\mathcal{F}_2)$. The *right-side gadget* is defined similarly except that the endpoint of each dangling edge must belong to V.

In the hardness proofs, we create generalized mating gadgets defined as follows, which is a generalization of the mating operation in [6]. In a *generalized mating gadget*, the variables of F are divided into four parts: *Sum-up* variables, *Fix-to-0* variables, *Fix-to-1* variables and a single *Dangling* variable. We assign F to each vertex of degree n labelled by a solid circle in Figure 2 in the following way: for each $1 \le a \le n$, if the ath variable is the Dangling variable, we let it correspond to the dangling edge. Otherwise we connect it to a vertex v_a of degree 2 labelled by a hollow square. Furthermore, if the ath variable is a Sum-up/Fix-to-0/Fix-to-1 variable, we assign a $[1, 0, 1]/[1, 0, 0]/[0, 0, 1]$ signature to v_a, and the gadget become well defined. In such constructions, the $p + q$ variables corresponding to $[1, 0]$ or $[0, 1]$ are always Sum-up variables. We remark that using $[1, 0]$ we may obtain the $[1, 0, 0] = [1, 0]^{\otimes 2}$ signature, and we construct a gadget of $[0, 0, 1]$ when needed.

2.2.2 Holographic Transformation

Let T be a binary signature, and we denote the two dangling edges corresponding to the input variables of it as a left edge and a right edge. Its value then can be written as a matrix $T = \begin{pmatrix} t_{00} & t_{01} \\ t_{10} & t_{11} \end{pmatrix}$, where t_{ij} is the value of T when the value of left edge is i and that of the right edge is j.

This notation is conducive to the efficient calculation of the gadget's value. Let us consider two binary signatures, T and P, with the right edge of T connected to the left edge of P. T and P now form a binary gadget. Subsequently, it can be demonstrated that the value of the resulting gadget is precisely TP, which represents the matrix multiplication of T and P.

For a signature f of arity n and a binary signature T, we use Tf/fT to denote the signature "f transformed by T", which is a signature of arity n obtained by connecting

the right/left edge of T to every dangling edge of f. For a set $\mathcal{F}$ of signatures, we also define $T\mathcal{F} = \{Tf | f \in \mathcal{F}\}$. Similarly we define $\mathcal{F}T$. The following theorem demonstrates the relationship between the initial and transformed problems:

▶ **Theorem 18** (Holographic Transformation[24, 3]). $Holant(\mathcal{F}|\mathcal{G}) \equiv_T Holant(\mathcal{F}T^{-1}|T\mathcal{G})$.

Let $H_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$. For a set of signatures $\mathcal{F}$, we use $\widehat{\mathcal{F}}$ to denote $H_2\mathcal{F}$. As $H_2^{-1} = \frac{1}{2}H_2$, by Theorem 18 we have:

$$\text{Holant}(\mathcal{F}|\mathcal{G}) \equiv_T \text{Holant}(\widehat{\mathcal{F}}|\widehat{\mathcal{G}}).$$

We additionally present the following fact as a lemma for future reference.

▶ **Lemma 19.** *For each $k \geq 1$, $\widehat{=_k} = [1,0,1,0,1,0,\dots]_k$. For example, $\widehat{=_1} = [1,0], \widehat{=_2} = [1,0,1], \widehat{=_3} = [1,0,1,0]$. Consequently, $\widehat{\mathcal{EQ}} = \{[1,0,1,0,1,0,\dots]_k | k \geq 1\}$ and for an integer $D \geq 1$, $\widehat{\mathcal{EQ}_{\leq D}} = \{[1,0,1,0,1,0,\dots]_k | 1 \leq k \leq D\}$.*

3 Matchgate signatures under variable permutations

In Section 3.1, we introduce the concept of the normalized matchgate signature, which simplifies the form of matchgate signatures. In Section 3.2, we prove that in polynomial time we may check whether a matchgate signature under a given permutation remains a matchgate signature. In Section 3.3, we characterize the permutable matchgate signatures in detail.

3.1 Normalize the matchgate

A matchgate signature is said to be *non-trivial* if it does not remain constant at 0. We say f is a *normalized signature* if $f(0\dots0) = 1$. For a normalized matchgate signature f of arity n and distinct $1 \leq b_1, \dots, b_k \leq n, 0 \leq k \leq n$, we define $F(b_1 \dots b_k) = f(\alpha)$ where $\alpha_{b_1} = \cdots = \alpha_{b_k} = 1$ and $\alpha_i = 0$ for each $1 \leq i \leq n, i \neq b_1, \dots, b_k$. For example, if the arity of f is 4, then $F(24) = f(0101)$ while $F() = f(0000) = 1$. We denote F as the *index expression* of f, and we also say F is a normalized matchgate signature without causing ambiguity.

This section presents the relationship between non-trivial matchgate signatures and normalized matchgate signatures, together with a property that normalized matchgate signatures have. The results in this section can be seen as a partial restatement of the results in [7].

▶ **Lemma 20.** *Each non-trivial matchgate signature g of arity n can be realized by a normalized matchgate signature f of arity n and $O(n)$ $[0,1,0]$ signatures, up to a constant factor.*

Proof. Since g is non-trivial, there exists $\beta \in \{0,1\}^n$ satisfying $g(\beta) \neq 0$. For each $\alpha \in \{0,1\}^n$, we let $f(\alpha) = g(\alpha \oplus \beta)/g(\beta)$. It can be verified that $f(0\dots0) = 1$ and f also satisfy MGI, which implies that f is a normalized matchgate signature. Then we can use the following gadget to realize $\frac{1}{g(\beta)}g$: for each $1 \leq i \leq n$ satisfying $\beta_i = 1$, we connect a $[0,1,0]$ signature to the ith variable of f. ◀

We denote f as the *normalization* of g in the above lemma. Suppose k is an integer, $b_1, \dots, b_{2k} \in \mathbb{N}^+$ and $b_1 < \cdots < b_{2k}$. $S = \{b_1, \dots, b_{2k}\}$ is said to be an *index set of size* $2k$. A *pairing* M of S is a partition of S whose components contain exactly 2 elements. In other

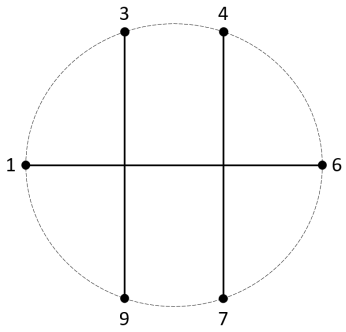


Figure 3 A visualization of the pairing $M = \{(1,6),(3,9),(4,7)\}$.

words, M can be seen as a perfect matching on the graph $(S, S \times S)$. In addition, suppose $a, b, c, d \in S$ and $a < b < c < d$. If M is a pairing of S and $ac, bd \in M$, then (ac, bd) is said to be a *crossing* in M. We use $c(M)$ to denote the number of crossings in M.

We also give a visualization of the definitions above. We draw all elements in S on a circle in a sequential order. Given a pairing M, we draw a straight line between the two elements in each pair belonging to M. A crossing in M is formed if and only if two of the straight lines form a crossing. See Figure 3 for an example. By the construction of the universal matchgate in [7], we have the following lemma.

▶ **Lemma 21.** *Suppose F is a normalized signature of arity n and of even parity. Then F is a matchgate signature if and only if for each integer $0 \le k \le n/2$ and distinct $1 \le b_1, \ldots, b_{2k} \le n$,*

$$F(b_1 \ldots b_{2k}) = \sum_{M:M \text{ is a pairing of } \{b_1,\ldots,b_{2k}\}} (-1)^{c(M)} \prod_{b_i b_j \in M} F(b_i b_j).$$

3.2 Permutation Check

In this section, we show that given a normalized matchgate signature F of arity n and a permutation π, we can decide whether F_π is a matchgate signature in polynomial time in n. To be precise, we only need to check whether F_π satisfy all the properties in Lemma 21 restricting to $k = 2$.

▶ **Theorem 22.** *Suppose F is a normalized matchgate signature and π is a permutation. If for each $1 \le a < b < c < d \le n$, $F_\pi(abcd) = F_\pi(ab)F_\pi(cd) - F_\pi(ac)F_\pi(bd) + F_\pi(ad)F_\pi(bc)$, then F_π is also a matchgate signature.*

The proof of Theorem 22 can be found in the full version. Here we present the proof sketch.

Proof Sketch. By Lemma 21, we only need to prove that for each integer $0 \le k \le n/2$ and distinct $1 \le b_1, \ldots, b_{2k} \le n$,

$$LHS \triangleq F_\pi(b_1 \ldots b_{2k}) = \sum_{M:M \text{ is a pairing of } \{b_1,\ldots,b_{2k}\}} (-1)^{c(M)} \prod_{b_i b_j \in M} F_\pi(b_i b_j) \triangleq RHS.$$

We prove this by induction. This statement is obviously true when $k = 1, 2$. Now suppose this statement is true for $k = 1, \ldots, p-1$, and we focus on the situation that $k = p$.

For convenience, we use a_i to denote $\pi(b_i)$ for each $1 \le i \le 2k$ in the following proof. As F is a matchgate signature, we already have that

$$LHS = F(a_1 \dots a_{2k}) = \sum_{M_\pi : M_\pi \text{ is a pairing of } \{a_1,\dots,a_{2k}\}} (-1)^{c(M_\pi)} \prod_{a_i a_j \in M_\pi} F(a_i a_j).$$

We begin by partitioning the set of all pairings. Suppose $S = \{b_1, \dots, b_{2k}\}$ and S_1, S_2 is a partition of S. Any pairing with no edges between S_1 and S_2 can be decomposed into two independent pairings on S_1 and S_2, respectively. By the induction hypothesis, each of these sub-pairings satisfies the desired property. Summing over some valid partitions, we derive a system of equations relating the partial sums on the left-hand side (LHS) and right-hand side (RHS). A careful analysis of these equations – exploiting the inductive structure and combinatorial symmetries – yields the desired global equality between the LHS and RHS summations. ◀

3.3 Characterizations of permutable matchgate signatures

By Theorem 22, we can use the following property to characterize permutable matchgate signature.

▶ **Corollary 23.** *Suppose F is a normalized matchgate signature of arity n. Then F is a permutable matchgate signature if and only if for each $1 \le a < b < c < d \le n$, $F(ab)F(cd) = F(ac)F(bd) = F(ad)F(bc)$.*

Proof. By Theorem 22, F is a permutable matchgate signature if and only if for any permutation $\pi \in S(n)$ and integers $1 \le a < b < c < d \le n$, $F_\pi(abcd) = F_\pi(ab)F_\pi(cd) - F_\pi(ac)F_\pi(bd) + F_\pi(ad)F_\pi(bc)$. Notice that $F(abcd) = F_\pi(\pi^{-1}(a)\pi^{-1}(b)\pi^{-1}(c)\pi^{-1}(d))$ also holds for arbitrary π, hence by taking different π, we have the following equation.

$$\begin{aligned} F(abcd) =& F(ab)F(cd) + F(ac)F(bd) - F(ad)F(bc) \\ =& F(ab)F(cd) - F(ac)F(bd) + F(ad)F(bc) \\ =& - F(ab)F(cd) + F(ac)F(bd) + F(ad)F(bc). \end{aligned}$$

which implies $F(ab)F(cd) = F(ac)F(bd) = F(ad)F(bc)$.

On the other hand, if $F(ab)F(cd) = F(ac)F(bd) = F(ad)F(bc)$ holds for arbitrary $1 \le a < b < c < d \le n$, then for any π, $F_\pi(abcd) = F(\pi(a)\pi(b))F(\pi(c)\pi(d)) = F_\pi(ab)F_\pi(cd) - F_\pi(ac)F_\pi(bd) + F_\pi(ad)F_\pi(bc)$. ◀

Using the description above, we are able to classify and characterize the normalized permutable matchgate signatures in the following way.

▶ **Lemma 24.** *For a normalized permutable matchgate signature F of arity $n \ge 4$, one of the following holds:*

1. *(Pinning type) For any distinct $1 \le a, b \le n$, we have $F(ab) = 0$.*
2. *(Parity type 1) There exist distinct $1 \le a, b, c, d \le n$, such that $F(ab)F(cd) \ne 0$.*
3. *(Parity type 2) There exist distinct $1 \le a, b, c \le n$, such that $F(ab)F(ac)F(bc) \ne 0$.*
4. *(Matching type) There exist distinct $1 \le a, b, c, d \le n$, such that $F(bc), F(bd), F(cd) = 0$, but $F(ab) \ne 0$.*

Proof. Suppose otherwise. As F is not of Pinning type, there exists $1 \le a, b \le n$ such that $F(ab) \ne 0$. Then for any distinct $1 \le c, d \le n$ and $c, d \ne a, b$, if $F(cd) \ne 0$, F is of Parity type 1, so we may assume $F(cd) = 0$.

As F is not of Parity type 1, $F(ac)F(bd) = F(ad)F(bc) = 0$. As F is not of Parity type 2, $F(ac)F(bc) = 0$ and $F(ad)F(bd) = 0$. Consequently, either $F(ac), F(ad) = 0$, or $F(bc), F(bd) = 0$. In either case F is of Matching type, a contradiction. ◀

If F is of Parity type 1, then there exist distinct $1 \leq a, b, c, d \leq n$, such that $F(ac)F(bd) = F(ad)F(bc) = F(ab)F(cd) \neq 0$, and we have $F(ab)F(ac)F(bc) \neq 0$, indicating that F is also of Parity type 2. Consequently, Parity type 1 and 2 can be concluded into a single type, denoted by *Parity type*, in which each signature satisfy the condition of Parity type 2.

Now we present our main theorems. We present the full proofs of them in Appendix A and B respectively. These proofs involve careful case analysis. We also note that $\mathscr{A} = \widehat{\mathscr{A}}$, and $\widehat{\mathscr{M}_P} - \widehat{\mathscr{A}}$ is exactly $\widehat{\mathscr{M}_P} - \mathscr{A} - \mathscr{P}$, consequently Theorem 26 characterize those signatures which are #P-hard on general graphs but computable on planar graphs in the setting of #CSP.

▶ **Theorem 25.** *Each permutable matchgate signature F' of arity n can be realized by a symmetric matchgate signature h and $O(n)$ symmetric binary matchgate signatures up to a constant factor in the following way.*

1. *If F' is of Pinning type after normalization, then $h = [1, 0, \ldots, 0]$ and $O(n)$ symmetric binary signatures $[0, 1, 0]$ can realize F'.*
2. *If F' is of Parity type after normalization, then $h = [1, 0, 1, 0, \ldots]$ or $[0, 1, 0, 1, \ldots]$ and the $O(n)$ symmetric binary signatures with the form $[1, 0, y]$ or $[0, 0, 1]$ can realize F'.*
3. *If F' is of Matching type, not of Pinning type and not of Parity type after normalization, then $h = [0, 1, 0, 0, \ldots]$ and the $O(n)$ symmetric binary signatures with the form $[1, 0, y]$ or $[0, 1, 0]$ can realize F'.*

▶ **Theorem 26.** *For each signature $F \in \mathscr{M}_P - \mathscr{A}$ of arity n, a symmetric signature $g \in \mathscr{M} - \mathscr{A}$ can be realized in the setting of Holant$(\{F\} \mid \{[1, 0], [1, 0, 1], [1, 0, 1, 0]\})$ as a planar left-side gadget.*

To summarize, Theorem 25 shows that permutable matchgate signatures are exactly symmetric matchgate signatures with possible binary modification on each variable. Furthermore, for each permutable matchgate signature that may lead to #P-hardness on general graphs, Theorem 26 gives a constructive way to symmetrize them in the setting of #CSP.

4 Conclusions

In this article, we prove a dichotomy for Pl-$\#R_D$-CSP, and transform the Pl-#CSP dichotomies into #CSP dichotomies on planar graphs. We present the sufficient and necessary condition for a matchgate signature f and a permutation π such that $\pi(f)$ is a matchgate signature as well, which can be checked in polynomial time. We also define the concept of permutable matchgate signatures, and characterize them in detail.

References

1 Jin-Yi Cai and Xi Chen. *Complexity Dichotomies for Counting Problems*. Cambridge University Press, 2017.

2 Jin-Yi Cai and Vinay Choudhary. Some results on matchgates and holographic algorithms. *Int J Software Informatics*, 1(1), 2007. URL: `http://www.ijsi.org/ch/reader/view_abstract.aspx?file_no=20073`.

3 Jin-Yi Cai and Vinay Choudhary. Valiant's Holant theorem and matchgate tensors. *Theoretical Computer Science*, 384(1):22–32, 2007. `doi:10.1016/j.tcs.2007.05.015`.

4 Jin-Yi Cai, Vinay Choudhary, and Pinyan Lu. On the theory of matchgate computations. *Theory of Computing Systems*, 45(1):108–132, 2009. doi:10.1007/s00224-007-9092-8.

5 Jin-Yi Cai and Zhiguo Fu. Holographic algorithm with matchgates is universal for planar #CSP over Boolean domain. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 842–855, 2017. doi:10.1145/3055399.3055405.

6 Jin-Yi Cai, Zhiguo Fu, and Shuai Shao. From Holant to quantum entanglement and back. In *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.22.

7 Jin-Yi Cai and Aaron Gorenstein. Matchgates revisited. *arXiv preprint arXiv:1303.6729*, 2013. arXiv:1303.6729.

8 Jin-Yi Cai, Pinyan Lu, and Mingji Xia. Holant problems and counting CSP. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 715–724, 2009. doi:10.1145/1536414.1536511.

9 Jin-Yi Cai, Pinyan Lu, and Mingji Xia. The complexity of complex weighted Boolean #CSP. *Journal of Computer and System Sciences*, 80(1):217–236, 2014. doi:10.1016/j.jcss.2013.07.003.

10 Nadia Creignou, Sanjeev Khanna, and Madhu Sudan. *Complexity classifications of Boolean constraint satisfaction problems*. SIAM, 2001.

11 Radu Curticapean and Mingji Xia. Parameterizing the permanent: Hardness for fixed excluded minors. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 297–307. SIAM, 2022. doi:10.1137/1.9781611977066.23.

12 Heng Guo and Tyson Williams. The complexity of planar Boolean #CSP with complex weights. *Journal of Computer and System Sciences*, 107:1–27, 2020. doi:10.1016/j.jcss.2019.07.005.

13 Pieter Kasteleyn. Graph theory and crystal physics. *Graph theory and theoretical physics*, pages 43–110, 1967.

14 Pieter W Kasteleyn. The statistics of dimers on a lattice: I. the number of dimer arrangements on a quadratic lattice. *Physica*, 27(12):1209–1225, 1961.

15 Pieter W Kasteleyn. Dimer statistics and phase transitions. *Journal of Mathematical Physics*, 4(2):287–293, 1963.

16 Susan Margulies and Jason Morton. Polynomial-time solvable #CSP problems via algebraic models and Pfaffian circuits. *Journal of Symbolic Computation*, 74:152–180, 2016. doi:10.1016/j.jsc.2015.06.008.

17 Boning Meng and Yicheng Pan. Dichotomies for #CSP on graphs that forbid a clique as a minor. *arXiv preprint arXiv:2504.01354*, 2025. doi:10.48550/arXiv.2504.01354.

18 Jason Morton. Pfaffian circuits. *arXiv preprint arXiv:1101.0129*, 2010.

19 Harold NV Temperley and Michael E Fisher. Dimer problem in statistical mechanics-an exact result. *Philosophical Magazine*, 6(68):1061–1063, 1961.

20 Dimitrios M Thilikos and Sebastian Wiederrecht. Killing a vortex. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1069–1080. IEEE, 2022. doi:10.1109/FOCS54457.2022.00104.

21 Leslie G Valiant. The complexity of computing the permanent. *Theoretical computer science*, 8(2):189–201, 1979. doi:10.1016/0304-3975(79)90044-6.

22 Leslie G Valiant. Quantum computers that can be simulated classically in polynomial time. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 114–123, 2001. doi:10.1145/380752.380785.

23 Leslie G Valiant. Expressiveness of matchgates. *Theoretical Computer Science*, 289(1):457–471, 2002. doi:10.1016/S0304-3975(01)00325-5.

24 Leslie G Valiant. Holographic algorithms. *SIAM Journal on Computing*, 37(5):1565–1594, 2008. doi:10.1137/070682575.

A Proof of Theorem 25

▶ **Lemma 27.** *If F is a normalized permutable matchgate signature of arity n of Parity type, then there exist a function $G : \{1, \ldots, n\} \to \mathbb{C}$ such that for any distinct $1 \le a, b \le n$, $F(ab) = G(a)G(b)$.*

Proof. As F is of Parity type, there exist distinct $1 \le a, b, c \le n$, such that $F(ab)F(ac)F(bc) \neq 0$. To avoid ambiguity, we use $\sqrt{F(ab)}$ to denote a specific complex number r_{ab} satisfying $r_{ab}^2 = F(ab)$. We also use $\sqrt{F(ab)F(ac)/F(bc)}$ to denote the complex number $\sqrt{F(ab)}\sqrt{F(ac)}/\sqrt{F(bc)}$. Let $G(a) = \sqrt{F(ab)F(ac)/F(bc)} \neq 0$, $G(b) = \sqrt{F(ab)F(bc)/F(ac)}$ and $G(c) = \sqrt{F(ac)F(bc)/F(ab)}$. For each $1 \le d \le n, d \neq a, b, c$, let $G(d) = F(ad)/G(a)$. Since $F(ab)F(ac)F(bc) \neq 0$, G is well-defined.

It is easy to verify that $F(ab) = G(a)G(b), F(ac) = G(a)G(c), F(bc) = G(b)G(c)$. For each $1 \le d \le n, d \neq a, b, c$,

$$G(a)G(d) = F(ad);$$

$$G(b)G(d) = \frac{F(ad)G(b)G(c)}{G(a)G(c)} = \frac{F(ad)F(bc)}{F(ac)} = \frac{F(bd)F(ac)}{F(ac)} = F(bd);$$

$$G(c)G(d) = \frac{F(ad)G(c)G(b)}{G(a)G(b)} = \frac{F(ad)F(bc)}{F(ab)} = \frac{F(cd)F(ab)}{F(ab)} = F(cd).$$

For any distinct $1 \le d, e \le n$ satisfying $d, e \neq a, b, c$,

$$G(d)G(e) = \frac{F(ad)F(be)}{G(a)G(b)} = \frac{F(de)F(ab)}{F(ab)} = F(de).$$ ◀

Now we analyze normalized permutable matchgate signatures of Matching type.

▶ **Lemma 28.** *Suppose F is a normalized permutable matchgate signature of arity n of Matching type. If F is not of Parity type, then there exists an integer $1 \le x \le n$ such that for any distinct $1 \le s, t \le n$, if $s, t \neq x$, $F(st) = 0$. Furthermore, if $2 \le k \le n/2$ and $1 \le b_1 < \cdots < b_{2k} \le n$, then $F(b_1 \ldots b_{2k}) = 0$.*

Proof. Suppose otherwise. Since F is of Matching type, there exist distinct $1 \le a, b \le n$ such that $F(ab) \neq 0$. There are three possible cases.

1. There exist distinct $1 \le s, t \le n, s, t \neq a, b$ such that $F(st) \neq 0$. It is obvious that a, b, s, t can serve as a certificate that F is of Parity type 1.
2. There exist distinct $1 \le s, t \le n, s, t \neq a, b$ such that $F(as)F(bt) \neq 0$. Again, a, b, s, t can serve as a certificate that F is of Parity type 1.
3. There exist $1 \le s \le n, s \neq a, b$ such that $F(as)F(bs) \neq 0$. In this case, a, b, s can serve as a certificate that F is of Parity type 2.

In each case, F is of Parity type, which is a contradiction.

By Lemma 21, if $2 \le k \le n/2$ and $1 \le b_1 < \cdots < b_{2k} \le n$, then $F(b_1 \ldots b_{2k}) = \sum_{M:M \text{ is a pairing of } \{b_1,\ldots,b_{2k}\}} (-1)^{c(M)} \prod_{b_i b_j \in M} F(b_i b_j) = 0$, since for each M of size greater than 2 there always exists $st \in M$ such that $s, t \neq x$. ◀

Proof of Theorem 25. Suppose f (or the index expression F of f) is a normalization of F'. For each case, we first realize F, then we analyze F' through Lemma 20.

If F is of Pinning type, by Lemma 21 $f = [1, 0, 0, \ldots, 0]$. By Lemma 20, F' can be realized by connecting a $[0, 1, 0]$ signature to some variables of F.

If F is of Parity type, by Lemma 27 there exists a function G such that for any distinct $1 \le a, b \le n$, $F(ab) = G(a)G(b)$. Then we can use the following gadget to realize F: for each $1 \le a \le n$, we connect a $[1, 0, G(a)]$ signature to the ath variable of $h' = [1, 0, 1, 0, \dots]$. By Lemma 20, F' can be realized by connecting a $[0, 1, 0]$ signature to some variables of F.

Now for some variables of h', they are connected to a $[1, 0, G(a)]$ signature, then a $[0, 1, 0]$ signature, where a is the index of the variable. For each such variable, we replace the $[1, 0, G(a)]$ with a $[0, 1, 0]$ and the $[0, 1, 0]$ with a $[G(a), 0, 1]$ respectively. The signature of the gadget remains the same after the replacement as $\begin{pmatrix} 1 & 0 \\ 0 & G(a) \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ G(a) & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} G(a) & 0 \\ 0 & 1 \end{pmatrix}$.

Now consider the gadget formed by $h' = [1, 0, 1, 0, \dots]$ and all the $[0, 1, 0]$ connecting to it. If there is an odd number of $[0, 1, 0]$ signatures, the signature of the gadget is $h = [0, 1, 0, 1, \dots]$. If there is an even number of $[0, 1, 0]$ signatures, the signature of the gadget is $h = [1, 0, 1, 0, \dots]$. For each $1 \le a \le n$, the binary signature connecting to the ath variable of h is either $[1, 0, G(a)]$ or $[G(a), 0, 1]$, which has the form $[1, 0, y]$ or $[0, 0, 1]$ up to a constant factor.

If F is of Matching type, not of Pinning type and not of Parity type, then by Lemma 28 there exists $1 \le x \le n$ such that the following holds: for any distinct $1 \le a, b \le n$, if $F(ab) \neq 0$, then $x \in \{a, b\}$. Let $G(x) = 1$ and $G(a) = F(ax)$ for each $1 \le a \le n, a \neq x$. It can be verified that the following gadget realize F: for each $1 \le a \le n, a \neq x$, we connect a $[1, 0, G(a)]$ signature to the ath variable of $h = [0, 1, 0, 0, \dots]$; we also connect a $[0, 1, 0]$ signature to the xth variable. By Lemma 20, F' can be realized by connecting a $[0, 1, 0]$ signature to some variables of F, which completes the proof. Besides, if there are two $[0, 1, 0]$ connecting to the xth variable of h, we can remove them without changing the signature of the gadget for future convenience. ◀

We denote the gadget in the above lemma as the *star gadget* ST_F of F, and h as the *central signature* h_F of ST_F. For each $1 \le a \le n$, all the binary signatures connecting to the ath variable of h in a line form a gadget, and the signature of the gadget is denoted by the *ath edge signature*.

B Proof of Theorem 26

We remark that $[1, 0], [1, 0, 1], [1, 0, 1, 0] \in \mathscr{M}$ and $\mathscr{M}$ is closed under gadget construction. Hence when proving the above theorem, we only need to ensure that the obtained g is symmetric and does not belong to $\mathscr{A}$.

Besides, such g must have one of the following forms:

▶ **Lemma 29** ([12]). *Suppose $g \in \mathscr{M} - \mathscr{A}$ and is symmetric. Then g has one of the following forms.*

1. $[0, 1, 0, ..., 0]_k, k \ge 3$;
2. $[0, ..., 0, 1, 0]_k, k \ge 3$;
3. $[1, 0, r], r^4 \neq 0, 1$;
4. $[1, 0, r, 0, r^2, ...]_k, k \ge 3, r^2 \neq 0, 1$;
5. $[0, 1, 0, r, 0, r^2, ...]_k, k \ge 3, r^2 \neq 0, 1$.

▶ Remark 30. In the following, we always construct the left-side gadget of Holant$(\{f\}|\{[1, 0], [1, 0, 1], [1, 0, 1, 0]\})$. For future convenience, whenever uv is an edge and both u and v are assigned f in a gadget, we actually mean that we replace uv with

uw, wv where w is assigned a $[1,0,1]$ signature in the gadget. Besides, if a gadget is formed by connecting two existing gadgets together, we also automatically replace the connecting edge uv with uw, wv, where w is assigned a $[1,0,1]$ signature. These operations would not change the signature of the gadget. Consequently, it can be verified that each obtained gadget always remains a left-side gadget in our following constructions.

If F is of Pinning type, we have $F \in \mathscr{A}$ as $[1,0,0,0,\dots], [0,1,0] \in \mathscr{A}$ and $\mathscr{A}$ is closed under gadget construction. Consequently, F is either of Parity type, or of Matching type and not of Parity type.

B.1 Parity type case

Suppose F is of Parity type. By Theorem 25, three kinds of binary signatures may connect to the central signature h_F in the star gadget ST_F, which are $[1,0,0], [0,0,1]$ and $[1,0,y], y \neq 0$ respectively. Suppose p $[1,0,0]$ and q $[0,0,1]$ are connected to h_F. Then we have $F = F' \otimes [1,0]^{\otimes p} \otimes [0,1]^{\otimes q}$ where $F' \in \mathscr{M}_P - \mathscr{A}$ is also of Parity type.

By assigning $n-p-q$ $[1,0]$ signature to F', we realize $[1,0]^{\otimes p} \otimes [0,1]^{\otimes q}$. By assigning $[1,0]^{\otimes p} \otimes [0,1]^{\otimes q}$ back to F again we can realize the F' signature in a planar way. Consequently, we may assume all the signatures connected to h_F has the form $[1,0,y], y \neq 0$ since we can eliminate all of the $[1,0]$s and $[0,1]$s by gadget construction. We also assume that for each $1 \leq a \leq n$, the ath variable of h_F is connected to the binary signature $[1,0,y_a]$.

Case 1: $h_F = [1,0,1,0,\dots]$. If $n = 2$, F is already symmetric and we are done. Now we assume $n \geq 3$. For distinct $1 \leq a, b \leq n$, if we connect a $[1,0]$ signature to each variable of F except for the ath and the bth one, we realize the $[1,0,y_a y_b]$ signature. If $(y_a y_b)^4 \neq 1$, $[1,0,y_a y_b] \notin \mathscr{A}$ and we are done.

Now we suppose for any distinct $1 \leq a, b \leq n$, $(y_a y_b)^4 = 1$. For any distinct $1 \leq a, b, c \leq n$, we have $y_a^8 = (y_a y_b \cdot y_a y_c / y_b y_c)^4 = 1$. If $y_a^4 = 1$ and $y_b^4 = -1$, again we are done since $(y_a y_b)^4 = -1 \neq 1$. If for each $1 \leq a \leq n$, $y_a^4 = 1$, then $[1,0,y_a] \in \mathscr{A}$. And since $[1,0,1,0,\dots] \in \mathscr{A}$ and $\mathscr{A}$ is closed under gadget construction, we have $F \in \mathscr{A}$, which is a contradiction.

Otherwise, for each $1 \leq a \leq n$, $y_a^4 = -1$. Let $\alpha = \sqrt{\mathrm{i}} = e^{2\pi \mathrm{i}/8}$, then $y_a = \pm\alpha, \pm\mathrm{i}\alpha$, and $[1,0,y_a y_b]$ is either $[1,0,\mathrm{i}]$ or $[1,0,-\mathrm{i}]$. Notice that each element in $\{\pm\alpha, \pm\mathrm{i}\alpha\}$ can become α by multiplying i or $-\mathrm{i}$ 0 to 3 times. Consequently, we may connect 0 to 3 $[1,0,\mathrm{i}]$ or $[1,0,-\mathrm{i}]$ to each variable of F to get a gadget ST, such that after replacing F with the gadget ST_F, each edge signature of the obtained star gadget ST'_F is exactly $[1,0,\alpha]$. It can be then verified that, the signature of ST, which is also the signature of ST'_F, is exactly $[1,0,\mathrm{i},0,-1,0,-\mathrm{i},0,1,\dots]_n$. We are done since $[1,0,\mathrm{i},0,-1,0,-\mathrm{i},0,1,\dots]_n \notin \mathscr{A}$ when $n \geq 3$.

Case 2:$h_F = [0,1,0,1,\dots]$. If $n = 2$, we suppose $F(1) = 1, F(2) = y$ despite the presence of a constant factor. In other words, $f = \begin{pmatrix} 0 & y \\ 1 & 0 \end{pmatrix}$. Because $F \notin \mathscr{A}$, $y^4 \neq 1$. By connecting the first variables of two F signatures to each other, we realize $[1,0,y^2]$. If $y^8 = (y^2)^4 \neq 1$, we are done. Otherwise, $y = \pm\alpha, \pm\mathrm{i}\alpha$. By connecting the second variables of three F signatures to $[1,0,1,0]$, as shown in Figure 4, we realize $[0,y^2,0,1]$, which is either $[0,\mathrm{i},0,1] \notin \mathscr{A}$ or $[0,-\mathrm{i},0,1] \notin \mathscr{A}$.

Otherwise, $n \geq 3$. For distinct $1 \leq a, b \leq n$, if we connect a $[1,0]$ signature to each variable of F except for the ath and the bth one, we realize the $\neq_2^{y_a, y_b} = \begin{pmatrix} 0 & y_b \\ y_a & 0 \end{pmatrix}$ signature. By connecting the second variable of $\neq_2^{y_a, y_b}$ to the ath variable of F, we construct a gadget GG_{F_a} whose signature is F_a.

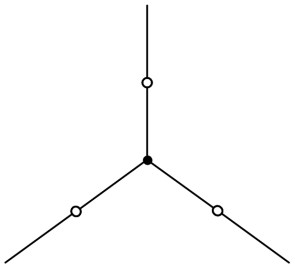


Figure 4 The construction of a gadget appears in the Case 2 of Parity type. The vertex of degree 3 represented by a solid circle is assigned $[1, 0, 1, 0]$, while each vertex of degree 2 represented by a hollowed circle is assigned F with the second variable connecting to $[1, 0, 1, 0]$.

Now we analyze the properties of F_a. By replacing F with the corresponding star gadget ST_F in GG_{F_a}, we obtain the gadget ST_a with central signature $h_F = [0, 1, 0, 1, \dots]$. For the ath variable of h_F, it is connected to a $[1, 0, y_a]$ signature, then a $\neq_2^{y_a, y_b}$ signature. We replace the $[1, 0, y_a]$ with $[0, 1, 0]$ and the $\neq_2^{y_a, y_b}$ with $[1, 0, y_b]$ respectively. As $\begin{pmatrix} 0 & y_b \\ y_a & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & y_a \end{pmatrix} = \begin{pmatrix} 0 & y_a y_b \\ y_a & 0 \end{pmatrix} = y_a y_b \begin{pmatrix} 1 & 0 \\ 0 & 1/y_b \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, the signature of the gadget remains the same up to a constant factor after the replacement.

Now consider the gadget formed by $h_F = [0, 1, 0, 1, \dots]$ and the $[0, 1, 0]$ connecting to the ath variable of it. The signature of the gadget is $h = [1, 0, 1, 0, \dots]$, which means that F_a belongs to Parity type Case 1. Consequently, we are done unless for each $1 \leq c \leq n, c \neq a$, $y_c^4 = 1$.

By connecting the first variable of $\neq_2^{y_a, y_b}$ to the bth variable of F, we construct a gadget GG_{F_b} whose signature is F_b. Similarly, we are done unless for each $1 \leq c \leq n, c \neq b$, $y_c^4 = 1$. As a result, the only case left is that for each $1 \leq c \leq n$, $y_c^4 = 1$. In this case, for each $1 \leq c \leq n$, $[1, 0, y_c] \in \mathscr{A}$. Since $[0, 1, 0, 1, \dots] \in \mathscr{A}$ and $\mathscr{A}$ is closed under gadget construction, $F \in \mathscr{A}$, a contradiction.

B.2 Matching type case

Suppose F is of Matching type and not of Parity type. By Theorem 25, two kinds of binary signatures may directly connect to the central signature h_F in the star gadget ST_F, which are $[1, 0, 0]$ and $[1, 0, y], y \neq 0$ respectively. Besides, the other variable of each of these signatures might also be connected to a $[0, 1, 0]$ signature. Suppose $p + q$ $[1, 0, 0]$ are connected to h_F directly and q of them are further connected to $[0, 1, 0]$. Then we have $F = F' \otimes [1, 0]^{\otimes p} \otimes [0, 1]^{\otimes q}$ where $F' \in \mathscr{M}_P - \mathscr{A}$ of arity $k = n - p - q$ is also of Matching type and not of Parity type.

When $k = 2$, the central signature of F' is $[0, 1, 0]$, and F is also of Parity type. Therefore, it can be demonstrated that $k \geq 3$. Let $ST_{F'}$ be the star gadget that realize F' as stated in Theorem 25, and $h_{F'}$ be the central signature. By Theorem 25, for each $1 \leq a \leq k$, exactly one of the following statements holds.

1. The ath variable of $h_{F'}$ is connected to a $[1, 0, y_a], y_a \neq 0$ signature. In this case we denote a as an upright index of F', and the ath variable of F' as an upright variable.
2. The ath variable of $h_{F'}$ is connected to a $[1, 0, y_a], y_a \neq 0$ signature, then a $[0, 1, 0]$ signature. In this case we denote a as a reversed index of F', and the ath variable of F' as a reversed variable.

Suppose the number of reversed indexes of F' is l. In the following, F' is examined in 4 possible cases based on l. In each case, we create generalized mating gadgets defined in Section 2.2.1.

In the subsequent analysis, we can see that two kinds of generalized mating gadgets would play an important role in each case: the Gadget 1 asks exactly one variable of F' to be Sum-up, while the Gadget 2 asks exactly two variables of F' to be Sum-up. All other upright variables are Fix-to-0 and all other reversed variables are Fix-to-1.

Also, it should be noted that the subsequent analysis does not include a detailed computation of the values of the signature of each gadget. Nevertheless, readers are encouraged to verify the results of these computations for themselves using the following observation:

Each variable corresponding to $[1,0]$ or $[0,1]$ is a Sum-up variable, and consequently does not contribute to the value of the signature. If a is an upright index, and the ath variable of F' is connected to $[1,0]$ or $[1,0,0]$, then the corresponding edge signature of F' does not contribute to the value of the gadget. Besides, the ath variable of the central signature $h_{F'}$ are pinned, while $h_{F'}$ remains the form $[0,1,0,0,\dots]$ after this pinning. Similarly, if a reversed variable of F' is connected to $[0,1]$ or $[0,0,1]$, the same statement holds. Furthermore, if a reversed variable is set to be the Sum-up variable in the generalized mating gadget, then the 2 $[0,1,0]$ signatures meet and have no effect to the value of the gadget.

Case 1: $l = 0$. Let $1 \le a, b, c \le k$ be distinct integers.

Gadget 1. Construct a generalized mating gadget. Let the ath variable be Dangling, the bth variable be Sum-up and all other variables of F' be Fix-to-0.

By Gadget 1, we realize the signature $[y_b^2, 0, y_a^2]$, and we are done unless $(y_a^2)^4 = (y_b^2)^4$. Similarly, by replacing b with c in the construction of the above gadget, we are done unless $(y_a^2)^4 = (y_c^2)^4$. Now we assume $(y_a^2)^4 = (y_b^2)^4 = (y_c^2)^4$.

Gadget 2. Construct a generalized mating gadget. Let the ath variable be Dangling, the bth and the cth variable be Sum-up and all other variables of F' be Fix-to-0.

By Gadget 2, we realize the signature $[y_b^2 + y_c^2, 0, y_a^2]$. Similarly by replacing a with b or c, we realize $[y_a^2 + y_c^2, 0, y_b^2]$ and $[y_a^2 + y_b^2, 0, y_c^2]$ respectively. As $(y_a^2)^4 = (y_b^2)^4 = (y_c^2)^4$, $|y_b^2 + y_c^2|, |y_a^2 + y_c^2|, |y_a^2 + y_b^2| \in \{0, \sqrt{2}|y_a^2|, 2|y_a^2|\}$ and one of them does not equal to 0. Without loss of generality let $|y_b^2 + y_c^2| \neq 0$, and we have $[y_b^2 + y_c^2, 0, y_a^2] \in \mathscr{M} - \mathscr{A}$.

Case 2: $l = 1$. Let $1 \le a, b, c \le k$ be distinct integers and a be the reversed index of F'.

Gadget 1. Construct a generalized mating gadget. Let the ath variable be Dangling, the bth variable be Sum-up and all other variables of F' be Fix-to-0.

By Gadget 1, we realize the signature $[y_a^2, 0, y_b^2]$, and we are done unless $(y_a^2)^4 = (y_b^2)^4$. Similarly, by replacing b with c in the construction of the above gadget, we are done unless $(y_a^2)^4 = (y_c^2)^4$. Now we assume $(y_a^2)^4 = (y_b^2)^4 = (y_c^2)^4$.

Gadget 2. Construct a generalized mating gadget. Let the ath variable be Dangling, the bth and the cth variable be Sum-up and all other variables of F' be Fix-to-0.

By Gadget 2, we realize the signature $[y_a^2, 0, y_b^2 + y_c^2]$. Similarly by replacing a with b or c, we realize $[y_a^2 + y_c^2, 0, y_b^2]$ and $[y_a^2 + y_b^2, 0, y_c^2]$ respectively. Similar to the analysis in Case 1, we are done.

Case 3: $l = 2$. Let $1 \le a, b, c \le k$ be distinct integers and b, c be reversed indices of F'.

We first realize the $[0,0,1]$ signature using F and $[1,0]$. By connecting a $[1,0]$ to each variable of F representing $[1,0]$, a $[1,0]$ to each upright variable of F' and a $[1,0]$ to the bth variable, we realize the $[0,1]^{\otimes q+1}$ signature. By making $\lfloor q/2 \rfloor$ self-loops on $[0,1]^{\otimes q+1}$, we realize either $[0,1]$ or $[0,0,1]$. If we realize $[0,1]$, we can realize $[0,0,1] = [0,1]^{\otimes 2}$ as well by tensor production.

Gadget 1. Construct a generalized mating gadget. Let the ath variable be Dangling, the bth variable be Sum-up, the cth variable be Fix-to-1 and all other variables of F' be Fix-to-0.

By Gadget 1, we realize the signature $[y_b^2, 0, y_a^2]$, and we are done unless $(y_a^2)^4 = (y_b^2)^4$. Similarly, by replacing b with c in the construction of the above gadget, we are done unless $(y_a^2)^4 = (y_c^2)^4$. Now we assume $(y_a^2)^4 = (y_b^2)^4 = (y_c^2)^4$.

Gadget 2. Construct a generalized mating gadget. Let the ath variable be Dangling, the bth and the cth variable be Sum-up and all other variables of F' be Fix-to-0.

By Gadget 2, we realize the signature $[y_b^2 + y_c^2, 0, y_a^2]$. Similarly by replacing a with b or c, we realize $[y_b^2, 0, y_a^2 + y_c^2]$ and $[y_c^2, 0, y_a^2 + y_b^2]$ respectively. Similar to the analysis in Case 1, we are done.

Case 4: $l \geq 3$. Let $1 \leq a, b, c \leq k$ be distinct reversed indices of F'.

We first realize the $[0, 0, 1]$ signature using F and $[1, 0]$. By connecting a $[1, 0]$ to each variable of F representing $[1, 0]$, a $[1, 0]$ to each upright variable of F' and a $[1, 0]$ to the cth variable, we realize the $[0, 1]^{\otimes q+l-1}$ signature. By making $\lfloor (q + l - 2)/2 \rfloor$ self-loops on $[0, 1]^{\otimes q+l-1}$, we realize either $[0, 1]$ or $[0, 0, 1]$. Again, if we realize $[0, 1]$, we can realize $[0, 0, 1] = [0, 1]^{\otimes 2}$ as well by tensor production.

Gadget 1. Construct a generalized mating gadget. Let the ath variable be Dangling, the bth variable be Sum-up, all other reversed variables of F' be Fix-to-1 and all other upright variables of F' be Fix-to-0.

By Gadget 1, we realize the signature $[y_a^2, 0, y_b^2]$, and we are done unless $(y_a^2)^4 = (y_b^2)^4$. Similarly, by replacing b with c in the construction of the above gadget, we are done unless $(y_a^2)^4 = (y_c^2)^4$. Now we assume $(y_a^2)^4 = (y_b^2)^4 = (y_c^2)^4$.

Gadget 2. Construct a generalized mating gadget. Let the ath variable be Dangling, the bth and the cth variable be Sum-up, all other reversed variables of F' be Fix-to-1 and all other upright variables of F' be Fix-to-0.

By Gadget 2, we realize the signature $[y_a^2, 0, y_b^2 + y_c^2]$. Similarly by replacing a with b or c, we realize $[y_b^2, 0, y_a^2 + y_c^2]$ and $[y_c^2, 0, y_a^2 + y_b^2]$ respectively. Similar to the analysis in Case 1, we are done.

Distributed Complexity of P_k-Freeness: Decision and Certification

Masayuki Miyamoto ✉
University of Tsukuba, Japan

Abstract

The class of graphs that do not contain a path on k nodes as an induced subgraph (P_k-free graphs) has rich applications in the theory of graph algorithms. This paper explores the problem of deciding P_k-freeness from the viewpoint of distributed computing.

For specific small values of k, we present the *first* CONGEST algorithms specified for P_k-freeness, utilizing structural properties of P_k-free graphs in a novel way. Specifically, we show that P_k-freeness can be decided in $\tilde{O}(1)$ rounds for $k = 4$ in the broadcast CONGEST model, and in $\tilde{O}(n)$ rounds for $k = 5$ in the CONGEST model, where n is the number of nodes in the network and $\tilde{O}(\cdot)$ hides a polylog(n) factor. The main technical contribution is a novel technique used in our algorithm for P_5-freeness to distinguish induced 5-paths from non-induced ones, which is potentially applicable to other induced subgraphs. This technique also enables the construction of a local certification of P_5-freeness with certificates of size $\tilde{O}(n)$. This improves $\tilde{O}(n^{3/2})$ by Bousquet and Zeitoun (TCS 2025), and is nearly optimal, given our $\Omega(n^{1-o(1)})$ lower bound on certificate size.

For general k, we establish the first CONGEST lower bound, which is of the form $n^{2-1/\Theta(k)}$. The $n^{1/\Theta(k)}$ factor is unavoidable, in view of the $O(n^{2-2/(3k+2)})$ upper bound by Eden et al. (Dist. Comp. 2022). Additionally, our approach yields the *first* superlinear lower bound on certificate size for local certification. This partially answers the conjecture on the optimal certificate size of P_k-freeness, asked by Bousquet et al. (arXiv:2402.12148).

Finally, we propose a novel variant of the problem called ordered P_k detection. We show that in the CONGEST model, the round complexity of ordered P_k detection is $\tilde{\Omega}(n)$ for $k \geq 5$, and in contrast, proving any nontrivial lower bound for ordered P_3 detection implies a strong circuit lower bound. As a byproduct, we establish a circuit-complexity barrier for $\Omega(n^{1/2+\varepsilon})$ *quantum* CONGEST lower bounds for induced 4-cycle detection. This is complemented by our $\tilde{O}(n^{3/4})$ quantum upper bound, which surpasses the classical $\tilde{\Omega}(n)$ lower bound by Le Gall and Miyamoto (ISAAC 2021).

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases subgraph detection, CONGEST model, local certification

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.51

Related Version *Full Version*: https://arxiv.org/abs/2410.20353

Acknowledgements The author would like to thank François Le Gall for discussions and for commenting on an earlier version of this paper. The author also acknowledges the anonymous reviewers for their helpful comments.

1 Introduction

1.1 Background

The subgraph detection problems in the distributed setting of limited communication bandwidth, including the CONGEST model, are interesting and well studied problems. For a given specific small pattern graph H (the description of H is known to all nodes of the network), the goal of H detection is to decide if the network G contains H as a subgraph (or an induced subgraph). If the network G contains H, at least one node outputs "yes", otherwise all nodes of the network output "no". (H detection is often referred as H freeness, where at least one node outputs "no" if the network contains H, and otherwise all nodes

© Masayuki Miyamoto;
licensed under Creative Commons License CC-BY 4.0

of the network output "yes". In this paper we abuse these two problems since they are essentially equivalent.) There are several variants previously considered in the literature: H listing requires to output all copies of H in G (each node outputs some copies of H and the union of all output lists are equivalent to the list of all copies of H in G). In recent years, significant progress in distributed subgraph detection has been made, and we now understand well the round complexities for detection of e.g., cliques and cycles. However, much less is known about induced paths. In the full version, we also summarize the previous results in this context.

Since it is known that in the CONGEST model, detecting non-induced k-paths can be done in $O(1)$ rounds for constant k [27], our focus in this paper is specifically on the induced k-path detection problem. Induced k-paths are particularly more important than non-induced paths because graphs that do not contain an induced k-path (often called P_k-free graphs) have numerous applications in algorithmic graph theory, particularly in the centralized setting. The absence of induced k-node paths imposes structural constraints on graphs, which can be leveraged to develop efficient algorithms for problems like independent set or coloring [5, 20, 22, 31, 23, 36]. Despite its importance, few results are known for P_k-freeness in the distributed setting. The only known result we are aware of is the result of Nikabadi and Korhonen [34], who explored the multicolored variant of the induced k-path detection, where each node is colored by an integer from $\{1, 2, \ldots, k\}$, and the goal is to detect an induced path on k different colors. They demonstrated that multicolored induced P_k detection requires $\Omega(n/\log n)$ rounds for any $k \geq 7$. However, their proof fails for P_k-freeness. To the best of our knowledge, there is no nontrivial result[1] for P_k-freeness in the CONGEST model beyond the general result by [14]: for any subgraph H on k nodes, there is a randomized CONGEST algorithm that solves induced H detection in $\tilde{O}(n^{2-\frac{2}{3k+1}})$ rounds.

Local certification of P_k-freeness

Another line of research focuses on local certification of P_k-freeness. In local certification, small labels called certificates are assigned to the nodes of the graph, and each node decides if the graph satisfies some property by using its local view. There are two important parameters, that is, the size of certificates and the locality of verification. In local certification of size s and locality ℓ, each node receives an $O(s)$-bit certificate from the external entity called the *prover*, and each node decides its binary output (*accept* or *reject*) only using the unique identifiers and certificates of its ℓ-hop neighbors. The scheme (with the locality $\ell = 1$) is introduced by [28] under the name of proof-labeling schemes, which is now a popular setting in the community of distributed computing:

▶ **Definition 1** (proof-labeling schemes). *A proof-labeling scheme (PLS_ℓ) of size s is a pair $(c, \mathcal{A})$ of functions c and $\mathcal{A}$ of the input graph $G = (V, E)$ such that*

- *for each node u, the function c assigns a bit string $c(u)$ of length s called the certificate of u;*
- *for each node u, the function $\mathcal{A}$, called the verification algorithm, depends on the certificates of ℓ-hop neighbors of u as well as the identifiers of these nodes, and decides the output of u. More precisely, let $N_\ell(u) = \{v_1, \ldots, v_d\}$ be the ℓ-hop neighbors of u, then*

$$\mathcal{A}(\mathsf{id}(u), c(u), \mathsf{id}(v_1), c(v_1), \ldots, \mathsf{id}(v_d), c(v_d))$$

is the output of u.

[1] For the trivial case of $k = 3$, P_3 detection can be done in $O(1)$ rounds.

Let $\mathcal{P}$ be a graph property. We say that there is a PLS_ℓ *that certifies $\mathcal{P}$ if there exists a verification algorithm $\mathcal{A}$ with locality ℓ (which outputs* Accept *or* Reject*) satisfies the following conditions.*

- *If G satisfies the property $\mathcal{P}$, there exists a certificate function c such that all nodes in G output* Accept*.*
- *If G does not satisfy the property $\mathcal{P}$, for any certificate function c, at least one node in G outputs* Reject*.*

When the locality of the verification algorithm is $\ell \geq \frac{k}{2}$, there is a trivial way to certify P_k-freeness: by assigning the list of IDs of neighbors as a certificate, each node try to find a P_k by looking edges adjacent to its $k/2$-hop neighbors. If there is a P_k, then its central node can now detect it. Thus, our interest lies in cases where the path length is long relative to the locality. Recently, Bousquet, Cook, Feuilloley, Pierron, and Zeitoun [2] studied this topic by analyzing the relationship among path length, certificate size, and locality of verification. They provided various nontrivial upper and lower bounds on certificate size (we discuss details in Section 1.2). While local certification of various subgraph-related problems has been studied (e.g., [3, 11, 18, 19]), the only known result for local certification of P_k-freeness prior to [2] was for $k = 4$: there is a PLS_1 that certifies P_4-freeness with $O(\log n)$ bits [17]. Moreover, [17] proved that every MSO_1 property Π can be certified with $O(\log^2 n)$ bits and locality 1 if all graphs satisfying Π have bounded clique-width. Since P_k-freeness can be expressed by MSO_1, this suggests that the difficulty gap between certifying P_4-freeness and P_5-freeness can be attributed to the fact that P_4-free graphs have bounded clique-width, whereas P_5-free graphs do not. After that, Bousquet and Zeitoun [4] constructed a PLS_1 that certifies P_5-freeness with $O(n^{3/2})$ bits.

1.2 Our results

1.2.1 Result 1: P_k-freeness for $k = 4$ and $k = 5$

We first show the following upper bounds for $k = 4$ and $k = 5$. Throughout the paper, we use "a randomized algorithm" as an algorithm that outputs the correct answer with probability at least $2/3$.

▶ **Theorem 2.** *There exists a randomized algorithm that solves P_4-freeness in the* broadcast CONGEST *model, running in $O\left(\frac{\log n}{\log \log n}\right)$ rounds.*

▶ **Theorem 3.** *There exists a randomized algorithm that solves P_5-freeness in the* CONGEST *model, running in $O\left(n \log^2 n\right)$ rounds.*

Technical challenges

As mentioned above, there were no nontrivial algorithms for P_k-freeness for $k \geq 4$ in the CONGEST model beyond the general $O(n^{2-2/(3k+2)})$ upper bound [14]. Moreover, the previous CONGEST algorithms for subgraph detection have been mostly focused on non-induced cases. Most techniques from these results seem to be useless for induced subgraph detecion. Typical examples are the *expander decomposition* applicable to cliques [8, 9], and the *BFS search with threshold* applicable to (non-induced) cycles [6, 18]. We thus need to exploit structural properties of P_4/P_5-free graphs, resulting our algorithms quite different from the other algorithms for subgraph detection. We believe that our approach hints new algorithms for other induced subgraphs.

We turn our attention to the framework of local certification. We get the following certification scheme for P_5-freeness.

▶ **Theorem 4.** *There is a* PLS_1 *that certifies* P_5*-freeness with certificates of size* $O(n \log n)$*.*

To obtain this result, we directly use the algorithm of Theorem 3. This is somewhat interesting, as CONGEST algorithms for subgraph freeness cannot be used for local certification of subgraph freeness in general (unless they are broadcast CONGEST algorithms). We also show that the size of our certificates is optimal, up to a subpolynomial factor:

▶ **Theorem 5.** *For any* $k \geq 4\ell + 1$*, any* PLS_ℓ *for* P_k*-freeness requires certificates of size* $\Omega\left(\frac{n}{e^{O(\sqrt{\log n})}}\right)$*.*

This result is proved via a combination of several known reduction techniques: we first reduce the nondeterministic three-party communication complexity of set-disjointness function to triangle freeness, which is then reduced to P_5-freeness.

Independent and concurrent works

After submitting the first draft of this paper, we learned about the following independent and concurrent works: Bousquet and Zeitoun [4] proved that there is a PLS_1 that certifies P_5-freeness with certificates of size $O(n^{3/2})$. Additionally, as noted in [4], Chaniotis, Cook, Hajebi, and Spirkl (independently and concurrently) obtained $\Omega(n^{1-o(1)})$ lower bound for P_5-freeness.

1.2.2 Result 2: P_k-freeness for general k

We provide the following lower bounds for P_k-freeness in the (randomized) CONGEST model using the two-party communication complexity, which is the standard framework for distributed subgraph detection problems.

▶ **Theorem 6.** *Let* d *be any positive integer.*

- *For* $d \leq 2$*,* P_k*-freeness for* $k \geq 11d$ *require* $\tilde{\Omega}(n^{2-1/d})$ *rounds in the randomized* CONGEST *model.*
- *For* $d \geq 3$*,* P_k*-freeness for* $k \geq 8d$ *requires* $\tilde{\Omega}(n^{2-1/d})$ *rounds in the randomized* CONGEST *model.*

The $n^{1/d} = n^{1/\Theta(k)}$ factor is unavoidable, in light of the upper bound of $n^{2-\Theta(1/k)}$ established by [14]. We will show these lower bounds using the standard reduction from two-party communication complexity. Although the framework used here is classic, there are several challenges that makes the proof highly non-trivial, which we discuss in detail later in Section 5.2.

Additionally, our constructions apply to the nondeterministic two-party communication setting, leading to the following.

▶ **Theorem 7.** *Let* ℓ *and* d *be positive integers.*

- *For* $d \leq 2$*, any* PLS_ℓ *for* P_k*-freeness requires certificates of size* $\tilde{\Omega}(n^{2-1/d})$ *for* $k \geq 4d\ell + 7d$*.*
- *For* $d \geq 3$*, any* PLS_ℓ *for* P_k*-freeness requires certificates of size* $\tilde{\Omega}(n^{2-1/d})$ *for* $k \geq 4d\ell + 4d$*.*

Comparison with the recent results by Bousquet et al. [2]

Let us now compare our results with those of Ref. [2], who considered local certification of P_k-freeness and obtained the following:

Table 1 Summary of our results and previous results on local certification of P_k-freeness. Here n denotes the number of nodes in the network.

Path length	Certificate size	Model	Reference
5	$O(n \log n)$	PLS_1	Thm 4
5	$O(n^{3/2})$	PLS_1	[4]
$4\ell + 1$	$\Omega(n^{1-o(1)})$	PLS_ℓ	Thm 5
$8\ell + 14$	$\tilde{\Omega}(n^{3/2})$	PLS_ℓ	Thm 7
$4d\ell + 4d$, $d \geq 3$	$\tilde{\Omega}(n^{2-1/d})$	PLS_ℓ	Thm 7
$3\ell - 1$	$O(n \log^3 n)$	PLS_ℓ	[2]
$4\ell + 3$	$\Omega(n)$	PLS_ℓ	[2]
$\lceil \frac{14\ell}{3} \rceil - 1$	$O(n^{3/2} \log^2 n)$	PLS_ℓ	[2]

▶ **Theorem 8** ([2]). *If the locality is $\ell \geq 1$, then:*

- *P_k-freeness for $k \leq 3\ell - 1$ can be certified with certificates of size $O(n \log^3 n)$;*
- *P_k-freeness for $k \leq \lceil \frac{14\ell}{3} \rceil - 1$ can be certified with certificates of size $O(n^{3/2} \log^2 n)$;*
- *P_k-freeness for $k \geq 4\ell + 3$ requires certificates of size $\Omega(n/\ell)$.*

They also conjectured the following.

▶ **Conjecture 9** ([2]). *For all $\alpha > 0$, the optimal size for local certification of $P_{\alpha\ell}$-free graphs with locality ℓ is of the form $\Theta(n^{2-1/f(\alpha)})$, for some unbounded increasing function f.*

We summarize our results and previous results in Table 1. For $\ell = 1$, the results of Bousquet et al. [2] provide nontrivial lower bound for P_7-freeness, and Bousquet and Zeitoun [4] provide nontrivial upper bound for P_5-freeness. In Theorems 4 and 5, we show the nearly optimal bound of P_5-freeness for $\ell = 1$.

For general k, Theorem 7 shows the first superlinear lower bounds, improving the previous linear lower bounds. Moreover, Theorem 7 partially answers Conjecture 9: f is at least linearly increasing.

1.2.3 Result 3: Ordered path detection and applications

We then define and study the following problem called ordered P_k detection.

▶ **Definition 10** (ordered P_k detection). *Each node of the graph has a color from $\{1, \ldots, k\}$, and the goal is to detect an induced path that consists of edges $\{(p_i, p_{i+1})\}_{i \in \{1,\ldots,k-1\}}$ on k nodes $\{p_i\}_{i \in \{1,\ldots,k\}}$ where p_i is colored by i.*

Motivation

The definition of ordered path detection may seem somewhat artificial; however, it can be motivated in a manner similar to that of multicolored path detection studied in [34]. Algorithms for the multicolored/ordered variants of these problems with color-coding techniques [1] are often used to address the standard version. For example, the state-of-the-art CONGEST algorithm for detecting $2k$-node cycles [6, 16] actually detects the ordered variant of $2k$-cycles. Consequently, lower bounds for the ordered variant reflect the limitations of algorithms that employ color-coding. Moreover, as demonstrated in Theorem 12, we will show that ordered path detection also provides some nontrivial insight for subgraph detection in the quantum CONGEST model, which is not restricted to algorithms using color-coding.

Contribution

We first prove the following lower bound for $k = 5$.

▶ **Theorem 11.** *Any randomized algorithm that solves ordered P_k detection for $k \geq 5$ requires $\tilde{\Omega}(n)$ rounds in the* CONGEST *model.*

We then focus on ordered P_3 detection. Our next finding is that, similarly to triangle detection, proving non-trivial lower bounds for ordered P_3 detection is difficult. This result also shows a circuit complexity barrier for induced C_4 detection.

▶ **Theorem 12** (Informal)**.** *For any constant $\varepsilon > 0$, proving any lower bound of the form $\Omega(n^{\varepsilon})$ for ordered 3-path detection in the* CONGEST *model or $\Omega(n^{1/2+\varepsilon})$ for induced C_4 detection in the quantum* CONGEST *model implies super-linear lower bounds on circuit complexity for an explicit family of boolean functions.*

Note that our circuit complexity barrier for induced C_4 detection is in the *quantum* CONGEST model [30]. We then complement this result by showing a nontrivial upper bound for induced C_4.

▶ **Theorem 13.** *In the quantum* CONGEST *model, induced C_4 detection can be solved in $O(n^{3/4})$ rounds with high probability.*

Previously, no nontrivial upper bound for induced C_4 detection was known, and our bound for induced C_4 detection beats a classical $\tilde{\Omega}(n)$ lower bound [29].

2 Preliminaries

We write $[n] = \{1, 2, \ldots, n\}$. We let $N(v)$ be the set of neighbors of v. For a graph $G = (V(G), E(G))$, we say that $H = (V(H), E(H))$ is a subgraph of G if there is an injective function $\phi : V(H) \to V(G)$ such that $(u, v) \in E(H) \Rightarrow (\phi(u), \phi(v)) \in E(G)$ for any pair of nodes $u, v \in V(H)$. We say that H is an induced subgraph of G if there is an injective function $\phi : V(H) \to V(G)$ such that $(u, v) \in E(H) \Leftrightarrow (\phi(u), \phi(v)) \in E(G)$ for any pair of nodes $u, v \in V(H)$. Let P_k be a path on k nodes. We say that G is P_k-free if G does not contain P_k as an induced subgraph. Throughout the paper, we assume that the input graph is connected, otherwise we can treat each connected component separately.

The CONGEST model and variants

In the CONGEST model [35], each node of the network $G = (V, E)$ has a distinct $O(\log n)$-bit identifier and can communicate with its neighbors in a synchronized manner. In each round each node (1) does some local computation and (2) sends an $O(\log n)$-bit message to each of its neighbors. In the initial state, each node only knows its own unique identifier (represented by $O(\log n)$ bits) and input, and the unique identifiers of its neighbors. The broadcast CONGEST model is a weaker model in the sense that each node can only broadcast a $O(\log n)$-bit message in each round. In the congested clique model [32], we allow all-to-all communication in each round. That is, the communication topology is always the n-node clique, and the input graph G is a subgraph of the clique. In the quantum CONGEST model [30], each node can send $O(\log n)$-qubit quantum message to each of its neighbors, instead of $O(\log n)$-bit classical message.

3 Algorithm for P_4-freeness

In this section we prove Theorem 2.

It is well known that a graph is P_4-free iff it is a cograph [10]. A cograph is defined as a graph that is constructed using the following rules:

- A single-node graph K_1 is a cograph;
- For cographs $G = (V_G, E_G)$ and $H = (V_H, E_H)$, its disjoint union $(V_G \uplus V_H, E_G \uplus E_H)$ is a cograph;
- For cographs $G = (V_G, E_G)$ and $H = (V_H, E_H)$, its join $(V_G \uplus V_H, E_G \uplus E_H \uplus (V_G \times V_H))$ is a cograph;

Using this characterization, we can construct a low-congestion spanning tree for any cograph as follows. A similar property on the spanning tree of cographs is also used in distributed (interactive) proofs for P_4-freeness [17, 33]. We use the following results on the balls into bins problem.

▶ **Lemma 14** (Balls into bins problem [21]). *Assume that there are n bins and n balls. Each ball uniform randomly selects one bin which it places into. Then, with probability at least $1 - 1/n$, all bins have at most $O\left(\frac{\log n}{\log \log n}\right)$ balls.*

▶ **Lemma 15.** *There exists a* broadcast CONGEST *algorithm that runs in $O(1)$ rounds and performs the following:*

- *If some node rejects, then G is not a connected cograph.*
- *Otherwise, with high probability, it outputs a rooted spanning tree of depth 2, where each node, except the root, has at most $O\left(\frac{\log n}{\log \log n}\right)$ children.*

Proof. We first reject if the diameter of the graph is not $O(1)$, since every connected cograph has diameter at most 2. This can be accomplished in $O(1)$ rounds by running a BFS search from the node with minimum identifier as the depth of the BFS tree provides a 2-approximation of the diameter. From the definition of cographs, a connected cograph $G = (V, E)$ is constructed from the join operation on two distinct node sets V_1 and V_2 where $V = V_1 \uplus V_2$ and $|V_1| \geq |V_2|$. Let r be the node with the maximum degree in G. Note that r can be found in $O(1)$ rounds since the diameter of the graph is constant. If $\deg(r) < n/2$, the algorithm rejects (if G is a connected cograph, then the maximum degree is at least $n/2$ as each node in V_2 has degree at least $|V_1| \geq n/2$). Consider the tree rooted at r where all neighbors of r are at depth 1. As below, we can construct the desired spanning tree with high probability if the graph is a connected cograph.

- Case 1: $r \in V_2$. Define $V_2' = V_2 \backslash (N(r) \cup \{r\})$. Observe that $V_1 \times V_2' \subseteq E$ and $|V_2'| \leq |V_1|$. Each $v \in V_2'$ uniform randomly selects one node from its neighbors as its parent in the tree. Now, each depth-1 node u (bin) is selected as the parent of v (ball) with probability at most $1/|V_2'|$ as $\deg(v) \geq |V_1|$. Then from Lemma 14, all depth-1 nodes have at most $O\left(\frac{\log n}{\log \log n}\right)$ children with high probability.
- Case 2: $r \in V_1$. Define $V_1' = V_1 \backslash (N(r) \cup \{r\})$. Since $\deg(r) \geq |V_1|$, we have

  $$|V_1'| = |V_1| - (\deg(r) - |V_2|) - 1 \leq |V_2|.$$

  Each node v in V_1' selects one of its neighbors as its parent uniform randomly. Now, each depth-1 node u (bin) is selected as the parent of v (ball) with probability at most $1/|V_2|$ as $\deg(v) \geq |V_2|$. Then from Lemma 14, all depth-1 nodes have at most $O\left(\frac{\log n}{\log \log n}\right)$ children with high probability. ◀

Proof of Theorem 2. Ref. [25] demonstrated a protocol in the randomized multiparty simultaneous messages model (also known as the distributed sketching model), where each node sends a message of size $O(\log n)$ to a referee. The referee can then determine if G is a cograph with high probability. We can simulate this protocol in the constructed depth-2 spanning tree, with the root r acting as the referee. Since each depth-1 node has at most $O\left(\frac{\log n}{\log\log n}\right)$ children, sending all messages to r is completed in $O\left(\frac{\log n}{\log\log n}\right)$ rounds. ◀

4 Algorithm and certification for P_5-freeness

In this section we give an overview of the proof strategy for Theorem 3 and Theorem 4. Since these theorems share the same proof strategy, we focus on our CONGEST algorithm for P_5-freeness (Theorem 3). Full proof of Theorem 4 will be found in the full version.

Our algorithm aims to efficiently detect whether a given graph is P_5-free, focusing on cases where the graph has a diameter of 2 or 3 (observe that if the diameter is at least 4, then it must not be P_5-free). We treat these cases separately due to their distinct properties.

4.1 Subgraph freeness in graphs with diameter two

Here we assume that the graph $G = (V, E)$ with n nodes and m edges has diameter 2. Let $r \in V$ be the node with maximum degree. Then $\deg(r) = \Delta = \Omega(m/n)$. Divide the node set $V = \{r\} \cup V_1 \cup V_2$ where V_i is the set of nodes whose distance to r is i. We first consider that every node u broadcasts the list of its neighbors in $O(n)$ rounds. After that r learns all the edges connected to $\{r\} \cup V_1$. In the remaining part we show that, with high probability, r can learn the edge set $E \cap (V_2 \times V_2)$ in another $O(n)$ rounds. Therefore, r can locally decide if the graph is P_5 free.

Assume that we divide V_2 into $\sqrt{m/n}$ subsets V_2^i for $i \in [\sqrt{m/n}]$ as follows (assuming $\sqrt{m/n}$ is an integer): each node $v \in V_2$ chooses an integer $i \in [\sqrt{m/n}]$ uniformly at random and joins V_2^i. Next, we label the set V_1 as $\{v_k | k \in [\Delta]\}$ where $\mathsf{ID}(v_j)$ is the k-th smallest among IDs of nodes in V_1. The partition of $V_2 = \{V_2^i\}$ and the label of V_1 nodes with respect to the order of IDs are informed to all the nodes in the network in $O(n)$ rounds. We use the following lemma.

▶ **Lemma 16** ([8]). *Consider a graph with $\bar{m}$ edges and $\bar{n}$ vertices. We generate a subset S by letting each vertex join S independently with probability p. Suppose that the maximum degree is $\Delta \leq \bar{m}p/20\log\bar{n}$ and $p^2\bar{m} \geq 400\log^2\bar{n}$. Then, with probability $1 - 1/n^c$ for sufficiently large c, the number of edges in the subgraph induced by S is at most $6p^2\bar{m}$.*

Here we set $\bar{n} = n$, $\bar{m} = m$, $p = \sqrt{n/m}$. Clearly, $p^2\bar{m} \geq 400\log^2\bar{n}$ holds. As $\Delta < n$, one can verify that $\Delta \leq \bar{m}p/20\log\bar{n}$ holds if $\bar{m} = m \geq 400n\log^2 n$. We assume that $m \geq 400n\log^2 n$ holds since otherwise a simple $O(m) = \tilde{O}(n)$-round algorithm suffices. Then, with high probability, $|E \cap V_2^i \times V_2^j| = O(p^2 m) = O(n)$ for all $i, j \in [\sqrt{m/n}]$. Consider the node $v_k \in V_1$ collects the edge set $E \cap V_2^i \times V_2^j$ for $k = i + (j-1)\sqrt{m/n}$. To do this, assume that $(u, v) \in E \cap V_2^i \times V_2^j$. Then there exists a node w such that $(u, w), (w, v_k) \in E$ since the diameter of the graph is 2. w knows the edge (u, v) because of the previous $O(n)$ round communication and thus w can send it to v_j. Since $|E \cap V_2^i \times V_2^j| = O(n)$, this is done in $O(n)$ rounds. Finally, v_k sends $E \cap V_2^i \times V_2^j$ to r in $O(n)$ rounds. Now we have established the following theorem.

▶ **Theorem 17.** *In graphs with diameter 2, for any subgraph H, (non-induced and induced) H-freeness can be solved in $O(n \log^2 n)$ rounds in the* CONGEST *model with high probability.*

It is noteworthy that this result does not hold when the diameter is greater than 2, since for any constant $\varepsilon > 0$, there is a pattern graph H such that H-freeness in graphs with diameter 3 requires $\Omega(n^{2-\varepsilon})$ rounds [15, 29].

4.2 P_5-freeness in graphs with diameter three

A High-Level Overview of Our Approach

In the case of a graph with diameter 3, the larger diameter necessitates more tricky definitions and case analysis, resulting in a more involved proof. We outline the main ideas of our proof here before giving details.

Let V_i be the set of nodes at distance i from r, for $i \in \{1, 2, 3\}$. The edges (except the ones incident to r) are partitioned as $E_{i,j} = E \cap (V_i \times V_j)$ for $i \leq j$. We employ a procedure $\mathcal{P}_{\mathsf{collect}}$ (Algorithm 1) that runs in $O(n)$ rounds, enabling r to gather $E_{1,1}$ and $E_{1,2}$.

Algorithm 1 $\mathcal{P}_{\mathsf{collect}}$.

r: the node with maximum degree.
T: the rooted spanning tree created by the BFS traversal from r.
Step 1 Each node broadcasts the list of its neighbors in $O(n)$ rounds.
Step 2 Consider the partition of V into $\sqrt{m/n}$ subsets V^i for $i \in [\sqrt{m/n}]$ as follows: Each node $v \in V$ chooses an integer $i \in [\sqrt{m/n}]$ uniformly at random, joins V^i and tells it to all other nodes so that all nodes in the graph learn the partition $\mathcal{V} = \{V^i\}_{i \in [\sqrt{m/n}]}$. The node r broadcasts the set of nodes that contains all nodes in V_1 in ascending order with respect to their IDs so that all nodes in the graph agree with $v_k \in V_1$ such that $\mathsf{ID}(v_k)$ is the k-th smallest ID in V_1.
Step 3 Define $\tilde{E}_{2,2}$ by the set of edges $(u, v) \in E_{2,2}$ satisfying at least one of the following:
1. $N(v_k) \cap (N(u) \cup N(v)) \neq \emptyset$ where $u \in V^i$ and $v \in V^j$ for the partition $\mathcal{V}$, and v_k is a neighbor of r such that the integer k is assigned in Step 2 for $k = i + (j-1)\sqrt{m/n}$,
2. $(N(u) \cup N(v)) \cap V_3 \neq \emptyset$.

We also define F_{bad} by the set of edges $(u, v) \in E_{2,2} \backslash \tilde{E}_{2,2}$ satisfying $N(u) \cap V_1 = N(v) \cap V_1$. Each node u tells the neighbors the list of its incident edges in $\tilde{E}_{2,2}$ and F_{bad}.
Step 4 For each edge (v_k, w), w sends the edge (u, v) between V^i and V^j to v_k if w received (u, v) in Step 1. After that, each $v_k \in V_1$ sends these edges between V^i and V^j to r. This takes $O(n)$ rounds since the number of edges between V^i and V^j is $O(n)$ w.h.p., due to Lemma 16.
Step 5 r computes $|E_{2,3}|, |\tilde{E}_{2,2}|, |F_{bad}|$. This is done by using the tree T. r then rejects if the following holds:
Condition 1 There is an edge in $E_{2,3} \cup \tilde{E}_{2,2}$ that is not sent to r.
For each edge $(u, v) \in E_{2,2} \backslash \tilde{E}_{2,2}$, u rejects if the following holds:
Condition 2 $N(u) \cap V_1 \neq N(v) \cap V_1$.
For each edge $(u, v) \in F_{bad}$, u rejects if the following holds:
Condition 3 There is a node $w \in N(u) \cap V_2$ such that $N(u) \cap V_1 \nsubseteq N(w)$.
For each edge $(u, v) \in E_{3,3}$, u rejects if the following holds:
Condition 4 $N(u) \cap V_2 \neq N(v) \cap V_2$.

For this procedure, we have the following.

▶ **Lemma 18** (Restated in Lemma 22). *If any of the conditions in $\mathcal{P}_{\mathsf{collect}}$ (Conditions 1-4) are met, then G contains an induced P_5.*

After the execusion of $\mathcal{P}_{\mathsf{collect}}$, if r detects any missing edge in $(E_{2,2}\backslash F_{bad}) \cup E_{2,3}$ – where F_{bad} is a carefully defined subset of $E_{2,2}$ (defined in Step 3 of $\mathcal{P}_{\mathsf{collect}}$) – it can safely conclude that the graph is not P_5-free. This is because (i) the invalidity of Condition 2 of $\mathcal{P}_{\mathsf{collect}}$ ensures that $E_{2,2}\backslash F_{bad} = \tilde{E}_{2,2}$, and (ii) the invalidity of Condition 1 of $\mathcal{P}_{\mathsf{collect}}$ ensures that all edges in $\tilde{E}_{2,2} \cup E_{2,3} = (E_{2,2}\backslash F_{bad}) \cup E_{2,3}$ have been sent to r. Therefore, if no node rejects in $\mathcal{P}_{\mathsf{collect}}$, r learns $E\backslash(F_{bad} \cup E_{3,3})$.

The following lemmas shows that, for any P_5 containing an edge from $F_{bad} \cup E_{3,3}$, detection is performed by nodes connected to the endpoints of these edges.

▶ **Lemma 19** (Restated in Lemma 23). *Assume that no node rejects in $\mathcal{P}_{\mathsf{collect}}$. If G contains an induced P_5 with at least one edge from F_{bad}, then there is a node that detects an induced P_5.*

▶ **Lemma 20** (Restated in Lemma 24). *Assume that no node rejects in $\mathcal{P}_{\mathsf{collect}}$. If G contains an induced P_5 with at least one edge from $E_{3,3}$, then there is a node that detects an induced P_5.*

For any P_5 composed solely of edges from $E\backslash(F_{bad} \cup E_{3,3})$, detection is performed by node r, which now has access to $E\backslash(F_{bad} \cup E_{3,3})$. The key challenge here is distinguishing between proper P_5s (induced by the entire edge set E) and improper P_5s (induced by $E\backslash(F_{bad} \cup E_{3,3})$ but not by E). We address this by having r count the improper patterns and subtract this count from the total number of P_5s it finds. As illustrated in Figure 1, there are 17 non-isomorphic patterns for possible improper P_5s. We introduce the concept of a "dangerous" induced subgraph. An induced copy of $H \in \mathcal{H}$ in the input network $G = (V, E)$ is *dangerous* iff it induces a P_5 in $E\backslash(F_{bad} \cup E_{3,3})$. The following lemma shows that the number of such dangerous patterns can be counted efficiently.

▶ **Lemma 21** (Restated in Lemma 25). *Let $\mathcal{H} = \{H_i\}_{i\in[17]}$ be a set of five-node graphs that contain P_5 as a (non-necessarily induced) subgraph as in Figure 1. Then, for any $H \in \mathcal{H}$, the number of induced copies of H in the graph that are dangerous can be counted (by the node r) in $O(n)$ rounds.*

This yields the correct count of proper P_5s, since the number of P_5s in G is now obtained by subtracting the number of dangerous patterns in G from the number of P_5s in $(V, E\backslash(F_{bad} \cup E_{3,3}))$.

4.2.1 Algorithm for P_5-freeness in graphs with diameter three

Assume that the diameter of the input graph (with n nodes and m edges) is 3 and $m \geq 400n\log^2 n$. Similarly to the previous case, we divide the node set into $V = \{r\} \cup V_1 \cup V_2 \cup V_3$ where V_i is the set of nodes whose distance to r is i. For all $i, j \in \{1, 2, 3\}$, $i \leq j$, we also define $E_{i,j}$ by the edge set between V_i and V_j. We use a subprotocol $\mathcal{P}_{\mathsf{collect}}$ described in Algorithm 1, and analyze it here.

▶ **Lemma 22.** *If any of the conditions in $\mathcal{P}_{\mathsf{collect}}$ (Conditions 1-4) are met, then G contains an induced P_5.*

Proof.

Condition 1 Assume that there is an edge in $E_{2,3}$ that is not sent to r in $\mathcal{P}_{\mathsf{collect}}$. Then for some i, j, k satisfying $k = i + (j-1)\sqrt{m/n}$, there exists an edge (u, v) between V^i and V^j such that no node in $N(u) \cup N(v)$ is connected to v_k. W.l.o.g., we assume that $u \in V_2$ and $v \in V_3$. Then there exists a node $w \in N(u) \cap V_1$ that is not connected to $v_k \in V_1$. $\{v_k, r, w, u, v\}$ induces a P_5.

Assume that there is an edge in $\tilde{E}_{2,2}$ that is not sent to r in $\mathcal{P}_{\mathsf{collect}}$. First, observe that if an edge (u,v) in $\tilde{E}_{2,2}$ satisfies the first condition of $\tilde{E}_{2,2}$, it is sent to r in $\mathcal{P}_{\mathsf{collect}}$. Assume that (u,v) satisfies the second condition of $\tilde{E}_{2,2}$ and is not sent to r in $\mathcal{P}_{\mathsf{collect}}$. Then w.l.o.g., there are two nodes $w \in N(u) \cap V_3$ and $x \in N(u) \cap V_1$. Furthermore, assume that $u \in V^i$, $v \in V^j$, and $k = i + (j-1)\sqrt{m/n}$. $\{v_k, r, x, u, w\}$ induces a P_5.

Condition 2 Suppose that, for some i, j, k satisfying $k = i + (j-1)\sqrt{m/n}$, there exists an edge $(u,v) \in E_{2,2} \backslash \tilde{E}_{2,2}$ between V^i and V^j satisfying $N(u) \cap V_1 \backslash N(v) \cap V_1 \neq \emptyset$. Then there exist a node $w \in V_1 \cap (N(u) \backslash N(v))$ that is not connected to $v_k \in V_1$. We can conclude that $\{v_k, r, w, u, v\}$ is an induced P_5.

Condition 3 Let $(u,v) \in F_{bad} \cap (V^i \times V^j)$. For a node $w \in (N(u) \cup N(v)) \cap V_2$ such that $N(u) \cap V_1 \nsubseteq N(w)$, if there is $y \in (N(u) \cap V_1) \backslash N(w)$, then $\{v_k, r, y, u, w\}$ is an induced P_5 for $k = i + (j-1)\sqrt{m/n}$.

Condition 4 Let $w \in (N(u) \backslash N(v)) \cap V_2$. Then there exists a node $x \in N(w) \cup V_1$. $\{r, x, w, u, v\}$ is a P_5. ◀

Now, assuming that $\mathcal{P}_{\mathsf{collect}}$ does not reject, the following properties hold for each edge $(u,v) \in F_{bad}$ between V^i and V^j.

Property 1 $N(u) \cap V_1 = N(v) \cap V_1$,

Property 2 $N(v_k) \cap (N(u) \cup N(v)) = \emptyset$ for $k = i + (j-1)\sqrt{m/n}$,

Property 3 $(N(u) \cup N(v)) \cap V_3 = \emptyset$,

Property 4 for each $w \in (N(u) \cup N(v)) \cap V_2$, $N(u) \cap V_1 = N(v) \cap V_1 \subseteq N(w)$.

▶ **Lemma 23.** *Assume that no node rejects in $\mathcal{P}_{\mathsf{collect}}$. If G contains an induced P_5 with at least one edge from F_{bad}, then there is a node that detects an induced P_5.*

Proof. We write $\{(p_i, p_{i+1})\}_{i \in \{1,2,3,4\}}$ be the four edges constructing some induced P_5. Here we assume that exactly one edge of them is bad. First, consider that $(p_2, p_3) \in F_{bad}$. Fix one node $v \in N(p_2) \cap N(p_3) \cap V_1$. Then from Property 1 and Property 3, p_4 is in V_2. Moreover, $p_4 \in N(v)$ from Property 4. Similarly, $p_1 \in V_2 \cap N(v)$. Now observe that v knows $\{(p_i, p_{i+1})\}_{i \in \{1,2,3,4\}}$ and v can detect the path.

We then consider that $(p_1, p_2) \in F_{bad}$. Similarly to the above case, we can assume that $p_3 \in V_2$. We distinguish the following cases.

Case 1: $p_4 \in V_2$. If $p_5 \in V_1$, p_3 can detect the path since p_3 knows $p_5 \notin N(p_2)$ and thus $p_5 \notin N(p_1)$ due to Property 1. If $p_5 \in V_3$, p_3 can detect the path due to Property 3. If $p_5 \in V_2$, for each node $v \in N(p_1) \cap V_1$, $v \in N(p_2) \cap N(p_3)$. If $(p_4, v) \in E$ or $(p_5, v) \in E$, v can detect the path. Otherwise $\{p_1, v, p_3, p_4, p_5\}$ is an induced P_5. p_3 can detect this path as follows. Since p_3 knows $(p_1, p_2) \in F_{bad}$ and $(v, P_5) \notin E$, p_3 can conclude that $(p_1, p_5) \notin E$ due to Property 4.

Case 2: $p_4 \in V_1$. If $p_5 \in V_1$, then p_3 can detect the path due to Property 1. Assume $p_5 \in V_2$. Fix arbitrary node $v \in N(p_1) \cap N(p_2) \cap N(p_3) \cap V_1$. If $p_5 \in N(v)$, then v can detect the path. Otherwise, p_3 knows that $p_5 \notin N(v)$ and $(p_2, p_5) \notin E$. p_3 can detect the path since p_3 can verify that $(p_1, p_5) \notin E$ due to Property 4.

Case 3: $p_4 \in V_3$. If $p_5 \in V_3$, then p_3 can detect the path since p_1, p_2 do not have a neighbor in V_3. Assume $p_5 \in V_2$. Fix arbitrary node $v \in N(p_1) \cap N(p_2) \cap N(p_3)$. If $p_5 \in N(v)$, then v can detect the path. Otherwise, p_3 knows that $p_5 \notin N(v)$. From Property 4, we have $p_5 \notin N(p_1) \cup N(p_2)$. p_3 can detect the path.

Therefore, any induced P_5 that contains exactly one edge from F_{bad} can be detected by some node.

Now we consider a path $\{(p_i, p_{i+1})\}_{i\in\{1,2,3,4\}}$ containing at least two edges from F_{bad}. If there are two consecutive bad edges in the path, $(p_i, p_{i+1}), (p_{i+1}, p_{i+2})$, there is a node $u \in V_1 \cap N(p_i) \cap N(p_{i+1}) \cap N(p_{i+2})$ from Property 1. u is also connected to p_{i+3} when $i \leq 2$ or p_{i-1} when $i \geq 2$. u can detect the path. The remaining case is the path containing two bad edges that are not consecutive. We have two distinct cases.

Case 4: $(p_1, p_2), (p_3, p_4) \in F_{bad}$. From Property 1 and Property 4, there is a node $u \in N(p_1) \cap N(p_2) \cap N(p_3) \cap N(p_4)$ who can detect the path.

Case 5: $(p_1, p_2), (p_4, p_5) \in F_{bad}$. This is similar to a subcase of Case 1. For each node $v \in N(p_1) \cap V_1$, $v \in N(p_2) \cap N(p_3)$. If $(p_4, v) \in E$ or $(p_5, v) \in E$, v can detect the path. Otherwise $\{p_1, v, p_3, p_4, p_5\}$ is an induced P_5. p_3 can detect this path as follows. Since p_3 knows $(p_1, p_2) \in F_{bad}$ and $(v, P_5) \notin E$, p_3 can conclude that $(p_1, p_5) \notin E$ due to Property 4. ◀

▶ **Lemma 24.** *Assume that no node rejects in $\mathcal{P}_{\mathsf{collect}}$. If G contains an induced P_5 with at least one edge from $E_{3,3}$, then there is a node that detects an induced P_5.*

Proof. We write $\{(p_i, p_{i+1})\}_{i\in\{1,2,3,4\}}$ be the four edges constructing some induced P_5. Consider that $(p_2, p_3) \in E_{3,3}$ for instance. $p_1, p_4 \in V_3$ since Condition 4 of $\mathcal{P}_{\mathsf{collect}}$ does not hold. Similarly, since $(p_3, p_4) \in E_{3,3}$ we have $p_5 \in V_3$. Therefore there exists $v \in N(p_1) \cap N(p_2) \cap N(p_3) \cap N(p_4) \cap N(p_5) \cap V_2$ that can detect the path. The analysis of the other cases is done in the same way – we can show that all path edges are from $E_{3,3}$. ◀

Given Lemmas 23 and 24, we can detect any induced P_5 that involves edges from $F_{bad} \cup E_{3,3}$.

Our focus now shifts to the detection of an induced P_5 solely composed of edges from $E \backslash (F_{bad} \cup E_{3,3})$. In this context, we need to ensure that any P_5 induced by $E \backslash (F_{bad} \cup E_{3,3})$ does not actually arise as an artifact of the removal of $F_{bad} \cup E_{3,3}$, but is genuinely induced by E. To address this, we consider all five-node induced subgraphs of G. As illustrated in Figure 1, there are exactly 17 distinct five-node patterns $\mathcal{H} = \{H_1, \ldots, H_{17}\}$, each containing a P_5 as a subgraph. See Figure 2 for the illustration.

We introduce the concept of a "dangerous" induced subgraph. An induced copy of $H \in \mathcal{H}$ in the input network $G = (V, E)$ is *dangerous* iff it induces a P_5 in $E \backslash (F_{bad} \cup E_{3,3})$.

▶ **Lemma 25.** *Let $\mathcal{H} = \{H_i\}_{i\in[17]}$ be a set of five-node graphs that contain P_5 as a (non-necessarily induced) subgraph as in Figure 1. Then, for any $H \in \mathcal{H}$, the number of induced copies of H in the graph that are dangerous can be counted (by the node r) in $O(n)$ rounds.*

Proof. We first note that there is no dangerous copy of $H \in \mathcal{H}$ including edges from both F_{bad} and $E_{3,3}$, since two endpoints of a bad edge do not have neighbors in V_3 due to Property 3 of F_{bad}.

We consider the case of H_1 in Figure 1, i.e., 5-node cycles C_5. Let

$$\{(c_1, c_2), (c_2, c_3), (c_3, c_4), (c_4, c_5), (c_5, c_1)\}$$

be five edges that form a dangerous C_5 where $(c_5, c_1) \in F_{bad}$ and $(c_i, c_{i+1}) \in E \setminus (F_{bad} \cup E_{3,3})$ for $i \in \{1, 2, 3, 4\}$. Then, from Property 1 and Property 3 of F_{bad}, we can assume that $c_2, c_4 \in V_2$. Moreover, from Property 4 all the nodes in $N(c_1) \cap V_1 = N(c_5) \cap V_1$ are neighbors of c_2 and c_4. Let u be the node in $N(c_1) \cap V_1 = N(c_5) \cap V_1$ with minimum identifier. Then u can detect the cycle as it knows all the neighbors of c_1, c_2, c_4, c_5. Each node counts the number of such cycles, and sends it to r. In this way, r can count the number of dangerous C_5 in the graph. Note that it is impossible to have $(c_5, c_1) \in E_{3,3}$ due to the fact that Condition 4 of $\mathcal{P}_{\mathsf{collect}}$ does not hold: If $(c_5, c_1) \in E_{3,3}$, then $c_2, c_4 \in V_3$. It contradicts the assumption that $(c_1, c_2) \notin E_{3,3}$.

For any other pattern graph from $\{H_2, \ldots, H_{17}\}$ illustrated in Figure 1, we can see that at least one node of the pattern is connected to at least three other nodes of the pattern who can detect the pattern. This completes the proof. ◀

Proof of Theorem 3. In $O(n)$ rounds, we can compute the diameter of the graph [24]. If the diameter exceeds 3, the network immediately rejects. If the diameter is two, we apply Theorem 17, which establishes that P_5-freeness can be verified in $O(n \log^2 n)$ rounds with high probability. So we focus on the case where the diameter is exactly 3.

First, if the number of edges in the graph satisfies $m \leq 400n \log^2 n$, the node r collects all edges in $O(m)$ rounds and decide if G is P_5-free. Otherwise, the network runs $\mathcal{P}_{\mathsf{collect}}$ in $O(n)$ rounds. Assuming $\mathcal{P}_{\mathsf{collect}}$ does not reject, r knows all the edges in $E \backslash (F_{bad} \cup E_{3,3})$ as $E_{2,2} \backslash \tilde{E}_{2,2} = F_{bad}$.

Next, we assume that there is no induced P_5 that contains at least one edge from $F_{bad} \cup E_{3,3}$ as such a P_5 can be detected by some node due to Lemma 23 and Lemma 24.

For each $H_i \in \mathcal{H}$, the node r counts the number of induced copies of H_i that are dangerous as in Lemma 25. Let $t(H_i)$ be the count and let $t = \sum_{i \in [17]} t(H_i)$ be the sum of them.

Finally, r counts the number of P_5's induced by $E \setminus (F_{bad} \cup E_{3,3})$. If it is equal to t, we can conclude that G is P_5-free as removing $F_{bad} \cup E_{3,3}$ from G increases the number of induced P_5's by t. Otherwise, if the count is strictly larger than t, G is not P_5-free. ◀

4.3 Local certification of P_5-freeness

We use the following standard technique.

▶ **Lemma 26.** *Let $G = (V, E)$ be an n-node connected graph, and T be a spanning tree rooted at arbitrary node t. For each $u \in V$, let $a(u) \in [\mathrm{poly}(n)]$ be an integer assigned to u. Then, there is a PLS_1 that computes $\sum_{u \in V} a(u)$ with certificates of size $O(\log n)$.*

Proof. Let T_u be a subtree of T rooted at u. The certificate to u consists of two $O(\log n)$-bit values b and $b(u)$. If u is a leaf, u rejects when $a(u) \neq b(u)$. If u is not a leaf, u rejects when $b(u) \neq a(u) + \sum_{v \in N(u) \cap T_u} b(v)$. The root node t rejects when $b \neq b(t)$. If no node rejects, $b = \sum_{u \in V} a(u)$. ◀

Proof of Theorem 4. We first describe the certificates and then explain the verification process and its correctness.

The certification comprises the following components, each represented by $O(n \log n)$ bits. We assume that $\sqrt{m/n}$ is an integer (otherwise, we use $\lceil \sqrt{m/n} \rceil$).

- Neighbors: For each node u, $\mathsf{Neighbors}(u)$ consists of a set of neighbors of u.
- Diameter: $\mathsf{Diameter}(u)$ consists of a BFS tree starting from u.
- SpanningTree: $\mathsf{SpanningTree}(u)$ describes a spanning tree rooted at the node with maximum degree, denoted r. Thus, it is identical to $\mathsf{Diameter}(r)$.
- Partition: $\mathsf{Partition}(u)$ provides a partition of nodes. The prover divides V into $\sqrt{m/n}$ subsets $\mathcal{V} = \{V^i\}_{i \in [\sqrt{m/n}]}$ so that the number of edges between V^i and V^j is $O(n)$ for all $i, j \in [\sqrt{m/n}]$. This partition exists due to Lemma 16. $\mathsf{Partition}(u)$ includes the partition $\mathcal{V}$ and a set containing all the neighbors of r in ascending order with respect to their IDs. Additionally, $\mathsf{Partition}(u)$ includes encoded values in $O(\log n)$-bit: $\tilde{m}_{2,2}$, m_{bad}, $m_{2,3}$, $m_{3,3}$, $\tilde{m}_{2,2}(u)$, $m_{bad}(u)$, $m_{2,3}(u)$, $m_{3,3}(u)$. These values are used to verify the size of edge subsets. Finally, $\mathsf{Partition}(u)$ includes, for each $H \in \mathcal{H}$, the numbers $c(H)$ and $c_u(H)$. These values are used to count the number of dangerous copy of H in the graph.

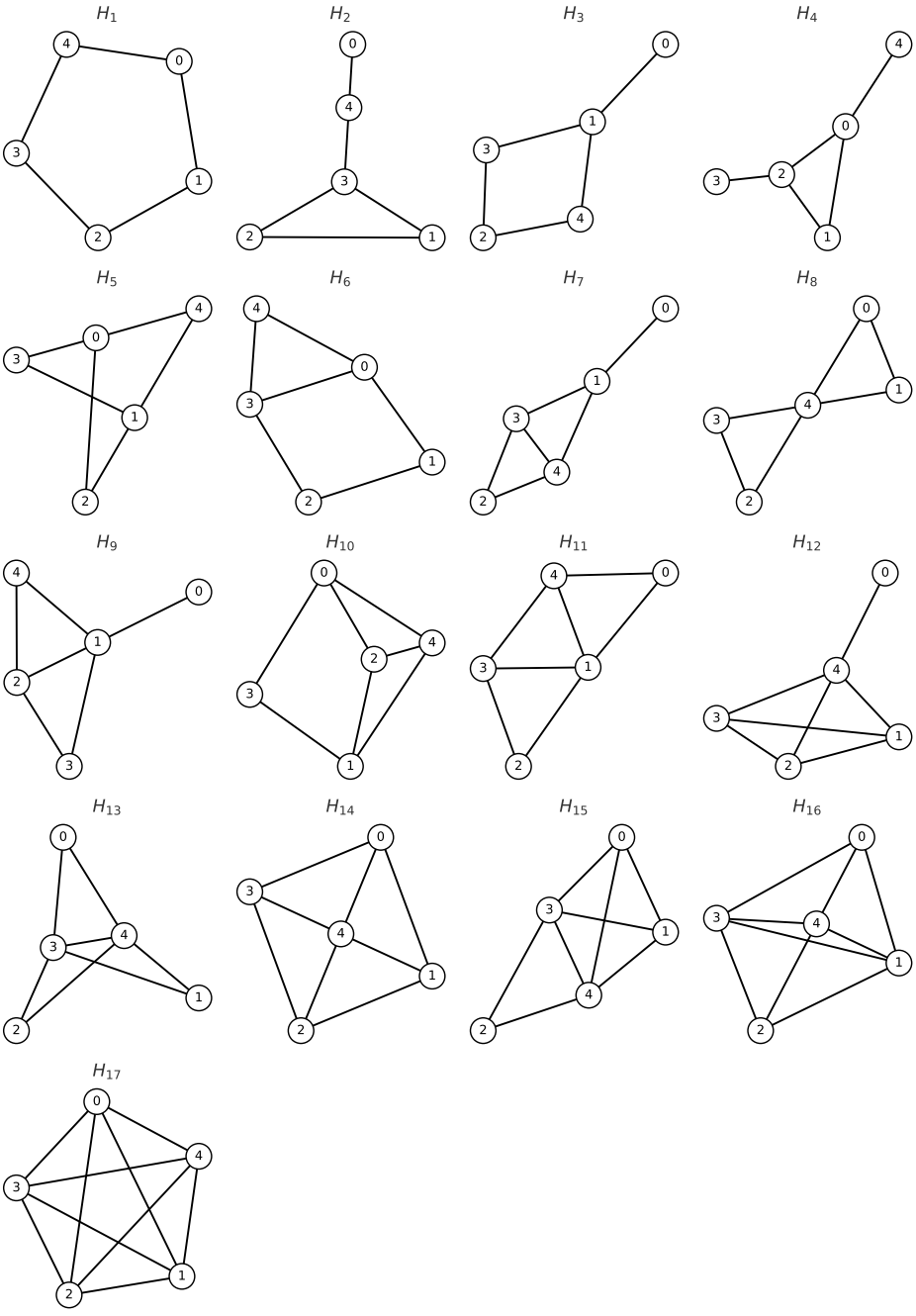


Figure 1 All different 5-node graphs that contain a P_5 as a subgraph.

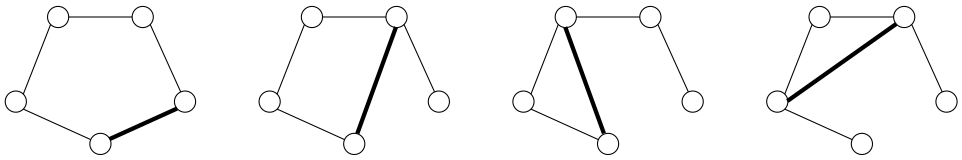


Figure 2 An illustration of H that contains exactly one bad edge considered in Lemma 25. Thick edges represent bad edges. For instance, an induced C_5 (the leftmost graph) contains exactly one edge from $F_{bad} \cup E_{3,3}$, and the remaining edges from $E \backslash (F_{bad} \cup E_{3,3})$, then removing $F_{bad} \cup E_{3,3}$ from the graph creates a new induced P_5.

- Edges: Let v_k be the neighbor of r whose degree is the k-th largest among all the neighbors of r for $k \in [m/n]$. $\mathsf{Edges}(v_k)$ then contains $O(n \log n)$-bit descriptions of all edges between V^i and V^j where $k = i + (j-1)\sqrt{m/n}$.

Verification

Diameter is used to certificate the graph diameter is at most 3. This is done by checking the structure of the trees as in [7].

Certifying the partition of nodes and edges. Each node u checks that the partition of nodes and the set representing all the neighbors of r in ascending order with respect to their IDs written in $\mathsf{Partition}(u)$, and the spanning tree written in $\mathsf{SpanningTree}(u)$ are the same as of its neighbors, so that all nodes agree the same partition, and the distance from r for all nodes. Now, given a partition of nodes in Partition and the spanning tree in SpanningTree, endpoints of each edge e determine which subset of

$$E_{1,1} \cup E_{1,2} \cup \tilde{E}_{2,2} \cup F_{bad} \cup E_{2,3} \cup E_{3,3}$$

the edge e belongs to. Next, each node computes the sizes of edge sets $|E_{1,1}|$, $|E_{1,2}|$, $|\tilde{E}_{2,2}|$, $|F_{bad}|$, $|E_{2,3}|$, $|E_{3,3}|$ using the protocol of Lemma 26 and the values written in Partition as follows: For example, we can compute $|E_{2,3}|$ by setting $b = |E_{2,3}|$, $b(u) = m_{2,3}(u)$, and $a(u)$ be the number of edges in $E_{2,3}$ incident to u.

Certifying the assigned edges. Let v_k be the neighbor of r that has the k-th minimum ID among all the neighbors of r. Since all nodes agree on the set representing all the neighbors of r in ascending order with respect to their IDs, all nodes know which is v_k. For each v_k, the validity of $\mathsf{Edges}(v_k)$ can be checked as follows. First, each v_k checks if the following holds.

1. $\mathsf{Edges}(v_k) \subseteq E \cap (V^i \times V^j)$ for $k = i + (j-1)\sqrt{m/n}$. v_k can check this since all nodes agree on the partition $\mathcal{V}$.
2. For each edge $e \in \mathsf{Edges}(v_k)$, at least one of its endpoints is at most distance 2 from v_k. v_k can check this since v_k knows nodes with distance 2 from v_k by looking at Neighbors of its neighbors.

If these conditions do not hold then v_k rejects. Each neighbor u of v_k rejects if $\mathsf{Edges}(v_k)$ contains non-edge that includes u' for some $u' \in N(u)$ by looking at $\mathsf{Neighbors}(u')$. This ensures that the edge set $\mathsf{Edges}(v_k)$ is indeed the set of edges between V^i and V^j such that at least one of the endpoints are a 2-hop neighbor of v_k. Each node then checks whether the four conditions described in $\mathcal{P}_{\mathsf{collect}}$ hold: r checks Condition 1, and each node u checks Condition 2,3, and 4 for each edge (u, v). If any condition holds, the network rejects. This part of verification ensures that r learns $E \backslash (F_{bad} \cup E_{3,3})$.

Detecting an induced P_5 containing $F_{bad} \cup E_{3,3}$. Assuming Conditions 1-4 of $\mathcal{P}_{\mathsf{collect}}$ are not met, some node can now detect an induced P_5 that contains at least one edge from $F_{bad} \cup E_{3,3}$ if exists, as in Lemma 23 and Lemma 24 since all necessary information from $\mathcal{P}_{\mathsf{collect}}$ is included in its certificate or the certificates of its neighbors.

Detecting an induced P_5 solely composed of $E \backslash (F_{bad} \cup E_{3,3})$. The final components of $\mathsf{Partition}(u)$ are used to count the number of dangerous H for $H \in \mathcal{H}$, as described in Lemma 25. This is accomplished using Lemma 26 as follows. For simplicity, let us consider the case of induced C_5 (H_1 in Figure 1). In our CONGEST algorithm, as in the proof of Lemma 25, for each such cycle, the node in V_1 with minimum identifier that is incident to the cycle detects it, and report the number of detected cycles to the node r through the BFS tree to compute the entire number of dangerous C_5's in the graph. Setting $b = c(H_1)$, $b(u) = c_u(H_1)$, and $a(u)$ be the number of dangerous C_5's detected by u in Lemma 26, we can certify it. r then determines the number of P_5's that increase by removing $F_{bad} \cup E_{3,3}$ (i.e., the number of dangerous copies of H for all $H \in \mathcal{H}$) and compares this with the number of induced P_5's in $(V, E \backslash (F_{bad} \cup E_{3,3}))$. Based on this comparison, r can decide if G is P_5-free. ◀

5 Lower bounds for P_k-freeness

5.1 Proof of Theorem 5

Our proof leverages a reduction from the three-party nondeterministic communication complexity of the set-disjointness problem in the number-on-forehead (NOF) model. In this model, each player sees the inputs of the other two players but not their own. The players communicate by writing bit strings on a shared blackboard, and the communication cost of the protocol is the total number of bits written.

As a first step, we use the reduction from set-disjointness to triangle-freeness established by Drucker et al.[13]. The key graph used in their reduction is a tripartite graph $G_{RS} = (A \cup B \cup C, E)$ where (1) $|A| = |B| = n$ and $|C| = n/3$ for arbitrary integer n; (2) G_{RS} contains $t = \frac{n^2}{e^{O(\sqrt{\log n})}}$ triangles $\mathcal{T} = \{T_1, \ldots, T_t\}$; (3) each edge of the graph belongs to exactly one triangle. This graph construction relates to the Ruzsa–Szemerédi problem[26] in extremal graph theory and has been well studied.

For $X_A, X_B, X_C \in \{0,1\}^t$, we define $G(X_A, X_B, X_C)$ as the graph obtained by removing edges from G_{RS} as follows. Let $T_i = (e_A, e_B, e_C)$ be a triangle such that e_A is an edge between B and C, e_B is an edge between A and C, and e_C is an edge between A and B. For $P \in \{A, B, C\}$, we remove the edge e_P iff the i-th bit of X_P is 0.

We then construct another graph $G^*(X_A, X_B, X_C)$ as follows. This construction is obtained by tweaking the construction in [12, 34]. Let $\overline{G}(X_A, X_B, X_C)$ be the complement graph of $G(X_A, X_B, X_C)$. For each node u in $\overline{G}(X_A, X_B, X_C)$, we have a triangle u_1, u_2, u_3 in $G^*(X_A, X_B, X_C)$. We add edges (u_i, v_i) for all $i \in \{1, 2, 3\}$. For each edge (u, v) in $\overline{G}(X_A, X_B, X_C)$, we add edges (u_i, v_j) for all $i, j \in \{1, 2, 3\}$ in $G^*(X_A, X_B, X_C)$. Additionally, there are two extra nodes x, y. x is connected to u_1 and u_2 for all $u \in V(\overline{G}(X_A, X_B, X_C))$. y is connected to u_2 and u_3 for all $u \in V(\overline{G}(X_A, X_B, X_C))$. Thus, the number of nodes in $G^*(X_A, X_B, X_C)$ is $O(n)$.

▶ **Lemma 27.** *$G^*(X_A, X_B, X_C))$ contains an induced P_5 if and only if X_A, X_B and X_C are not disjoint.*

Proof. Suppose that there is an index i such that the i-th bit of X_A, X_B and X_C are all 1. Then we have a triangle T_i in the graph $G(X_A, X_B, X_C)$ and thus have an independent set $\{u, v, w\}$ of size 3 in $\overline{G}(X_A, X_B, X_C)$. $\{u_1, x, v_2, y, w_3\}$ induces P_5 in $G^*(X_A, X_B, X_C)$.

Conversely, assume that $G^*(X_A, X_B, X_C)$ contains an induced P_5. Let us focus on an independent set of size 3 in the path. The set does not contain x and y at the same time, since each other node is connected at least one of x and y. Moreover, the set cannot contain x, as otherwise the other two nodes are written u_3 and v_3, but it is impossible since u_3 and v_3 are always connected. A similar contradiction occurs if the set contains y. Therefore three nodes in the set are written u_1, v_2, w_3. From the construction u, v, w in $\overline{G}(X_A, X_B, X_C)$ is an independent set. This means that X_A, X_B and X_C are not disjoint. ◀

We are now ready to prove Theorem 5.

Proof of Theorem 5. Suppose that P_5-freeness can be certified with certificates of size s. We construct a non-deterministic communication protocol for set-disjointness with communication complexity $O(ns)$. Suppose that Alice, Bob, and Charlie, who receive the inputs X_A, X_B, and X_C, respectively, simulate $G^*(X_A, X_B, X_C)$ as follows:

- For all $u \in A$ in $\overline{G}(X_A, X_B, X_C)$, Alice simulates nodes u_1, u_2, u_3 in $G^*(X_A, X_B, X_C)$.
- For all $u \in B$ in $\overline{G}(X_A, X_B, X_C)$, Bob simulates nodes u_1, u_2, u_3 in $G^*(X_A, X_B, X_C)$.
- For all $u \in C$ in $\overline{G}(X_A, X_B, X_C)$, Charlie simulates nodes u_1, u_2, u_3 in $G^*(X_A, X_B, X_C)$.
- All players simulates x and y.

Since the incident edges of simulated nodes by each player are independent from its input, each player can construct the inputs of its simulated nodes locally. Then the players simulate the certification for P_5-freeness. Since the certificate size is s and each player simulates $O(n)$ nodes, the length of the certificates for each player is $O(ns)$. In order to determine the outputs of the simulated nodes, each player writes its certificates to the shared blackboard. From Lemma 27, the players can compute set-disjointness of size $\frac{n^2}{e^{O(\sqrt{\log n})}}$. We now obtain $s = \Omega\left(\frac{n}{e^{O(\sqrt{\log n})}}\right) = \Omega(n^{1-o(1)})$ as desired by the nondeterministic multiparty communication complexity of set-disjointness in the number-on-forehead model [37]. To extend the lower bound for larger locality, we just replace edges incident to x and y by longer paths. Thus we can increase the locality by 1 at the cost of increasing the path length by 4. ◀

5.2 Lower bounds for P_k-freeness: general k

Technical challenges and the general proof idea for Theorem 6

We use a standard reduction from two-party set-disjointness function for the proof of Theorem 6 (and Theorem 7). Alice and Bob jointly construct some graph which depends on their inputs so that the constructed graph is P_k-free iff the output of some function (in our case, the set-disjointness function) is 1.

The main challenge in this proof is to ensure that the cut of the constructed graph (i.e., the number of edges between nodes held by Alice and those held by Bob) is as sparse as possible. This was not an issue in the proof of Theorem 5, which uses the non-deterministic communication complexity of the NOF model or in the proof of the lower bound given in [2], which uses counting arguments. In fact, the constructions used in these proofs do not achieve a sparse cut, making them unsuitable for the proof of Theorem 6 that holds for the CONGEST model.

Fortunately, in the simplest case where $d \geq 3$, we can utilize the construction from Ref. [29] with a slight modification, taking advantage of the graph structure that holds for all $d \geq 3$. However, for the cases where $d = 1$ and $d = 2$, the construction from Ref. [29], which

is applicable to induced k-cycles for every $k \geq 4$, fails for induced k-paths. This is intuitively because we lose several symmetric properties of induce k-cycles by removing one edge from the cycle. We thus construct two other graph families from scratch. The main challenge in establishing these constructions is that both edges and non-edges must be carefully chosen to avoid creating unexpected induced k-paths. Our proof requires a detailed analysis to ensure this, as the general structure valid for $d \geq 3$ does not apply for $d \leq 2$.

6 Ordered path detection and applications

In this section, we address the problem of detecting an ordered P_k. In this problem, each node of the graph is assigned a color from $\{1, \ldots, k\}$, and the objective is to detect an induced path $\{(p_i, p_{i+1})\}_{i \in \{1,\ldots,k-1\}}$ on k nodes $\{p_1, \ldots p_k\}$, where each node p_i is colored with i. Our first result is that detecting an ordered P_5 is already challenging, unlike the case of P_k-freeness where we only know nontrivial lower bound for $k \geq 11$ as in Theorem 6.

▶ **Theorem 28.** *Any randomized algorithm that solves ordered P_k detection for $k \geq 5$ requires $\tilde{\Omega}(n)$ rounds in the* CONGEST *model.*

The proof of this theorem leverages a reduction from two-party communication complexity, with a construction that is notably simpler compared to the proof of Theorem 6. The detailed proof is provided in the full version.

Our next insight regarding ordered path detection is the connection between proving non-trivial lower bounds for ordered P_3 detection and a long-standing open problem in circuit complexity.[2] Moreover, this result has implications for the detection of induced C_4 in the quantum CONGEST model. Specifically, the following problem has remained unresolved for decades:

▶ **Open Problem 1.** *Construct an explicit family of boolean functions $f_n : \{0,1\}^n \to \{0,1\}$ such that there exist constants $\delta_1, \delta_2 > 0$ such that any family of circuits $\{C_n\}_{n \in \mathbb{N}}$, where each circuit C_n is depth $O(n^{\delta_1})$ and consists of $O(n^{1+\delta_2})$ gates with constant fan-in and fan-out, cannot compute $\{f_n\}_{n \in \mathbb{N}}$.*

The following theorem formalizes this relationship:

▶ **Theorem 29** (The formal version of Theorem 12)**.** *Let $\varepsilon > 0$ be a constant. Then proving $\Omega(n^{\varepsilon})$ lower bound for ordered P_3 detection in the* CONGEST *model or $\Omega(n^{1/2+\varepsilon})$ lower bound for induced C_4 detection in the quantum* CONGEST *model solves Open Problem 1.*

▶ Remark 30. One might think that ordered P_3 detection can be solved in $O(1)$ rounds as the case of P_3-freeness. However, it is easily shown that the round complexity of ordered P_3 detection in the CONGEST model is actually at least the round complexity of triangle detection in the congested clique model: Let G' be the graph resulting by deleting all edges in the input graph G between a node colored by 1 and a node colored by 3, and replacing non-edges between them by edges. Then any CONGEST algorithm for ordered P_3 detection is simulated in the congested clique model with the input graph G' where each ordered P_3 in G corresponds to a multicolored triangle in G'. Note that multicolored triangle detection can be reduced to (the standard) triangle detection in this model [34].

[2] This kind of connection is already known for several subgraphs such as triangles and 6-cycles [15, 14].

Proof sketch of Theorem 29. The first part of the statement builds on the known hardness results for triangle detection (and counting) established by [14]. We demonstrate that ordered P_3 detection is at least as difficult as certain variants of triangle counting or ordered P_3 counting in high-conductance graphs. The full proof is deferred to the full version.

For the second part, we use the framework of distributed quantum search [30]. Consider a function $f : X \to \{0, 1\}$ defined on some finite set X, and assume a T-round CONGEST algorithm allows a fixed node u to output $f(x)$ with constant probability, given input $x \in X$. In the quantum CONGEST model, we can find an element $x \in X$ such that $f(x) = 1$ (if such an element exists) with constant probability in $\tilde{O}(\sqrt{|X|} \cdot T)$ rounds.

Now, assume that ordered P_3 detection can be solved in T rounds in the CONGEST model. We can show that induced C_4 detection can be solved in $\tilde{O}(\sqrt{n} \cdot T)$ rounds in the quantum CONGEST model. The statement then follows from the first part of the theorem, as an $\Omega(n^{1/2+\varepsilon})$ lower bound for induced C_4 detection implies an $\Omega(n^{\varepsilon})$ lower bound for ordered P_3 detection.

Specifically, let u be an arbitrary node, and define $M(u) = \{v : \mathrm{dist}(u, v) = 2\}$. Consider the subgraph $G[N(u) \cup M(u)]$ induced by $N(u)$ and $M(u)$. Nodes in $M(u)$ are colored with color 2, while nodes in $N(u)$ randomly choose their color from 1 and 3. Note that $G[N(u) \cup M(u)]$ contains an ordered P_3 (p_1, p_2, p_3) if G contains an induced C_4 (u, p_1, p_2, p_3). Since p_1 and p_3 randomly choose their colors, the probability that (p_1, p_2, p_3) is properly colored is constant. Therefore, in $O(T)$ rounds, we can determine if u belongs to an induced C_4. By using distributed quantum search for a function $f : V \to \{0, 1\}$, which outputs $f(u) = 1$ if and only if u belongs to an induced C_4, we can solve induced C_4 detection in $\tilde{O}(\sqrt{n} \cdot T)$ rounds. ◀

References

1. Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM (JACM)*, 42(4):844–856, 1995. doi:10.1145/210332.210337.
2. Nicolas Bousquet, Linda Cook, Laurent Feuilloley, Théo Pierron, and Sébastien Zeitoun. Local certification of forbidden subgraphs. *arXiv preprint arXiv:2402.12148*, 2024. doi:10.48550/arXiv.2402.12148.
3. Nicolas Bousquet, Laurent Feuilloley, and Théo Pierron. Local Certification of Graph Decompositions and Applications to Minor-Free Classes. In *25th International Conference on Principles of Distributed Systems (OPODIS 2021)*, pages 22:1–22:17, 2022. doi:10.4230/LIPIcs.OPODIS.2021.22.
4. Nicolas Bousquet and Sébastien Zeitoun. A subquadratic certification scheme for p5-free graphs. *Theoretical Computer Science*, page 115091, 2025. doi:10.1016/J.TCS.2025.115091.
5. Christoph Brause, Ingo Schiermeyer, Přemysl Holub, Zdeněk Ryjáček, Petr Vrána, and Rastislav Krivoš-Belluš. 4-colorability of P_6-free graphs. *Electronic Notes in Discrete Mathematics*, 49:37–42, 2015.
6. Keren Censor-Hillel, François Le Gall, and Dean Leitersdorf. On distributed listing of cliques. In *Proceedings of the 39th Symposium on Principles of Distributed Computing*, pages 474–482, 2020. doi:10.1145/3382734.3405742.
7. Keren Censor-Hillel, Ami Paz, and Mor Perry. Approximate proof-labeling schemes. *Theoretical Computer Science*, 811:112–124, 2020. doi:10.1016/J.TCS.2018.08.020.
8. Yi-Jun Chang, Seth Pettie, and Hengjie Zhang. Distributed triangle detection via expander decomposition. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 821–840. SIAM, 2019. doi:10.1137/1.9781611975482.51.
9. Yi-Jun Chang and Thatchaphol Saranurak. Improved distributed expander decomposition and nearly optimal triangle enumeration. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 66–73, 2019. doi:10.1145/3293611.3331618.

10 Derek G Corneil, Helmut Lerchs, and L Stewart Burlingham. Complement reducible graphs. *Discrete Applied Mathematics*, 3(3):163–174, 1981. doi:10.1016/0166-218X(81)90013-5.

11 Pierluigi Crescenzi, Pierre Fraigniaud, and Ami Paz. Trade-Offs in Distributed Interactive Proofs. In *33rd International Symposium on Distributed Computing (DISC 2019)*, pages 13:1–13:17, 2019. doi:10.4230/LIPICS.DISC.2019.13.

12 Mina Dalirrooyfard, Thuy Duong Vuong, and Virginia Vassilevska Williams. Graph pattern detection: Hardness for all induced patterns and faster non-induced cycles. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 1167–1178, 2019. doi:10.1145/3313276.3316329.

13 Andrew Drucker, Fabian Kuhn, and Rotem Oshman. On the Power of the Congested Clique Model. In *Proceedings of the 2014 ACM symposium on Principles of distributed computing (PODC 2014)*, pages 367–376, 2014. doi:10.1145/2611462.2611493.

14 Talya Eden, Nimrod Fiat, Orr Fischer, Fabian Kuhn, and Rotem Oshman. Sublinear-time distributed algorithms for detecting small cliques and even cycles. *Distributed Computing*, pages 1–28, 2022.

15 Orr Fischer, Tzlil Gonen, Fabian Kuhn, and Rotem Oshman. Possibilities and impossibilities for distributed subgraph detection. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures*, pages 153–162, 2018. doi:10.1145/3210377.3210401.

16 Pierre Fraigniaud, Maël Luce, Frederic Magniez, and Ioan Todinca. Even-cycle detection in the randomized and quantum congest model. In *Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing (PODC 2024)*, pages 209–219, 2024.

17 Pierre Fraigniaud, Frédéric Mazoit, Pedro Montealegre, Ivan Rapaport, and Ioan Todinca. Distributed certification for classes of dense graphs. In *Proceedings of the 37th International Symposium on Distributed Computing (DISC 2023)*, pages 20:1–20:17, 2023. doi:10.4230/LIPICS.DISC.2023.20.

18 Pierre Fraigniaud, Pedro Montealegre, Ivan Rapaport, and Ioan Todinca. A meta-theorem for distributed certification. *Algorithmica*, 86(2):585–612, 2024. doi:10.1007/S00453-023-01185-1.

19 Pierre Fraigniaud and Dennis Olivetti. Distributed detection of cycles. *ACM Transactions on Parallel Computing (TOPC)*, 6(3):1–20, 2019. doi:10.1145/3322811.

20 Peter Gartland and Daniel Lokshtanov. Independent Set on P_k-Free Graphs in Quasi-Polynomial Time. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 613–624. IEEE, 2020.

21 Gaston H Gonnet. Expected length of the longest probe sequence in hash code searching. *Journal of the ACM (JACM)*, 28(2):289–304, 1981. doi:10.1145/322248.322254.

22 Andrzej Grzesik, Tereza Klimošová, Marcin Pilipczuk, and Michał Pilipczuk. Polynomial-time algorithm for maximum weight independent set on P_6-free graphs. *ACM Transactions on Algorithms (TALG)*, 18(1):1–57, 2022. doi:10.1145/3414473.

23 Chính T Hoàng, Marcin Kamiński, Vadim Lozin, Joe Sawada, and Xiao Shu. Deciding k-colorability of P_5-free graphs in polynomial time. *Algorithmica*, 57:74–81, 2010. doi:10.1007/S00453-008-9197-8.

24 Stephan Holzer and Roger Wattenhofer. Optimal distributed all pairs shortest paths and applications. In *Proceedings of the 2012 ACM symposium on Principles of distributed computing*, pages 355–364, 2012. doi:10.1145/2332432.2332504.

25 Jarkko Kari, Martín Matamala, Ivan Rapaport, and Ville Salo. Solving the induced subgraph problem in the randomized multiparty simultaneous messages model. In *International Colloquium on Structural Information and Communication Complexity (SIROCCO 2015)*, pages 370–384, 2015. doi:10.1007/978-3-319-25258-2_26.

26 János Komlós and Miklós Simonovits. Szemeredi"s regularity lemma and its applications in graph theory, 1995.

27 Janne H. Korhonen and Joel Rybicki. Deterministic Subgraph Detection in Broadcast CONGEST. In *Proceedings of the 21st International Conference on Principles of Distributed Systems (OPODIS 2017)*, pages 4:1–4:16, 2018. doi:10.4230/LIPIcs.OPODIS.2017.4.

28 Amos Korman, Shay Kutten, and David Peleg. Proof labeling schemes. *Distributed Computing*, 22(4):215–233, 2010. doi:10.1007/S00446-010-0095-3.

29 François Le Gall and Masayuki Miyamoto. Lower Bounds for Induced Cycle Detection in Distributed Computing. In *32nd International Symposium on Algorithms and Computation (ISAAC 2021)*, pages 58:1–58:19, 2021. doi:10.4230/LIPICS.ISAAC.2021.58.

30 François Le Gall and Frédéric Magniez. Sublinear-time quantum computation of the diameter in CONGEST networks. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing (PODC 2018)*, pages 337–346, 2018. URL: https://dl.acm.org/citation.cfm?id=3212744.

31 Daniel Lokshantov, Martin Vatshelle, and Yngve Villanger. Independent set in P_5-free graphs in polynomial time. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on discrete algorithms*, pages 570–581. SIAM, 2014. doi:10.1137/1.9781611973402.43.

32 Zvi Lotker, Elan Pavlov, Boaz Patt-Shamir, and David Peleg. Mst construction in o (log log n) communication rounds. In *Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 94–100, 2003. doi:10.1145/777412.777428.

33 Pedro Montealegre, Diego Ramírez-Romero, and Ivan Rapaport. Compact distributed interactive proofs for the recognition of cographs and distance-hereditary graphs. In *International Symposium on Stabilizing, Safety, and Security of Distributed Systems (SSS 2021)*, pages 395–409, 2021. doi:10.1007/978-3-030-91081-5_26.

34 Amir Nikabadi and Janne Korhonen. Beyond distributed subgraph detection: Induced subgraphs, multicolored problems and graph parameters. In *25th International Conference on Principles of Distributed Systems (OPODIS 2021)*, volume 217, 2022. doi:10.4230/LIPIcs.OPODIS.2021.15.

35 David Peleg. *Distributed computing: a locality-sensitive approach*. SIAM, 2000.

36 Bert Randerath and Ingo Schiermeyer. 3-Colorability $\in P$ for P_6-free graphs. *Discrete Applied Mathematics*, 136(2-3):299–313, 2004.

37 Anup Rao and Amir Yehudayoff. Simplified lower bounds on the multiparty communication complexity of disjointness. In *30th Conference on Computational Complexity (CCC 2015)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.CCC.2015.88.

Local Routing on Ordered Θ-Graphs

André van Renssen
The University of Sydney, Australia

Shuei Sakaguchi
The University of Sydney, Australia

Abstract

The problem of locally routing on geometric networks using limited memory is extensively studied in computational geometry. We consider one particular graph, the ordered Θ-graph, which is significantly harder to route on than the Θ-graph, for which a number of routing algorithms are known. Currently, no local routing algorithm is known for the ordered Θ-graph.

We prove that, unfortunately, there does not exist a deterministic memoryless local routing algorithm that works on the ordered Θ-graph. This motivates us to consider allowing a small amount of memory, and we present a deterministic $O(1)$-memory local routing algorithm that successfully routes from the source to the destination on the ordered Θ-graph. We show that our local routing algorithm converges to the destination in $O(n)$ hops, where n is the number of vertices. To the best of our knowledge, our algorithm is the first deterministic local routing algorithm that is guaranteed to reach the destination on the ordered Θ-graph.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Ordered Θ-graph, Local routing, Computational geometry

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.52

Related Version *Full Version*: https://arxiv.org/abs/2506.16021

Funding *André van Renssen*: This research was partially funded by the Australian Government through the Australian Research Council (project number DP240101353).

1 Introduction

A *geometric network* $G = (V, E)$ is a weighted graph where each vertex $v \in V$ is a point in the Euclidean plane $\mathbb{R}^2$ and each edge $(u, v) \in E$ is a straight line segment, connecting $u \in V$ and $v \in V$, weighted by the Euclidean distance $|uv|$. We define the *weighted distance* $\delta_G(u, v)$ between two vertices u and v on G to be the sum of the weights of the edges along the weighted shortest path from u to v in G.

A subgraph $H = (V, E')$ of a geometric network $G = (V, E)$ is called a c-spanner of G, if, for every pair of vertices $s, t \in V$, we have $\delta_H(s, t) \leq c \cdot \delta_G(s, t)$, where $c \in \mathbb{R}_{\geq 1}$. Here, G is called the *underlying network* of H. The smallest constant c such that H is a c-spanner of G is called the *spanning ratio* or *stretch factor* of H. We consider the situation where the underlying network G is the *complete Euclidean graph* of V – i.e., a c-spanner H of G approximates the Euclidean distance $|st| = \delta_G(s, t)$ between any pair of points $s, t \in V$ by a constant factor c. The spanning ratio of a class of spanners $\mathcal{G}$ is the spanning ratio of the worst-case instance $G \in \mathcal{G}$ maximizing the spanning ratio. See the textbook by Narasimhan and Smid [15] and the survey by Bose and Smid [9] for a comprehensive overview of spanners and their open problems.

The study of geometric spanners is closely related to the design and analysis of efficient routing algorithms that forward a message between a pair of vertices. Given a source vertex s and a target vertex t on a graph G, a *routing algorithm* aims to find a short path from s to t. If the information of the entire graph can be known and kept track during routing, several

© André van Renssen and Shuei Sakaguchi;
licensed under Creative Commons License CC-BY 4.0

classical path-finding algorithms, including Dijkstra's algorithm [11] and the Bellman-Ford-Moore algorithm [14], can be used on spanners to find a short path. However, when we have restricted knowledge of the graph and limited memory, the routing problem becomes more challenging.

A routing algorithm is called *h-local* if it only knows the h-neighbourhood of the current vertex u and the information of the destination t, where the h-neighbourhood of u is the set of vertices that can be reached from u by taking at most h distinct edges in the graph. A routing algorithm is called *m-memory* if a memory of size m is stored with the message of routing.[1]

Two major classes of local routing algorithms have been actively researched in the literature: (1) deterministic 1-local memoryless routing algorithms and (2) deterministic 1-local $O(1)$-memory routing algorithms. Borrowing the notation used by Bose et al. [2], these two classes of local routing algorithms can be formalised as follows:

- A deterministic 1-local memoryless routing algorithm on a graph $G = (V, E)$ is a *routing function* $f : V \times V \times \mathcal{P}(V) \to V$, where $\mathcal{P}(\cdot)$ denotes the power set. We express the parameters of the function as $f(u, t, N(u))$, where $u \in V$ is the "current" vertex during routing (i.e., u currently holds the message we want to route from the source to the destination), $t \in V$ is the destination vertex, and $N(u) \subseteq V$ is the 1-neighbourhood of u (which simply means the set of immediate neighbours of u). The output of the routing function with the given parameters, $v = f(u, t, N(u)) \in N(u)$, is the neighbour of u to which the message is forwarded.
- A deterministic 1-local $O(1)$-memory routing algorithm on a graph $G = (V, E)$ is a routing function $g : V \times V \times \mathcal{P}(V) \times \mathcal{I} \to V$, which is similar to the routing function f above, except the parameters of the function $g(u, t, N(u), i)$ has an additional parameter $i \in \mathcal{I}$, modelling some information stored in the constant-sized memory of g that can be used to make routing decisions.

A routing algorithm performed on a graph H is called *c-competitive* with respect to the underlying network G if the total weight of the path traveled by the routing algorithm between any pair of vertices in H is at most c times the total weight of the shortest path in G [4]. The minimum possible value c for which the given routing algorithm is c-competitive with respect to G is called the *routing ratio* of the routing algorithm in H.

The Θ-graph, introduced independently by Clarkson [10] and Keil [13], is a geometric network studied extensively in the literature. Given a point set $V \subset \mathbb{R}^2$, the Θ_k-graph $G = (V, E)$ is defined and constructed as follows: For each vertex $u \in V$, we let equally spaced k rays emerge from u, resulting in k cones, each with the aperture $\theta = \frac{2\pi}{k}$. For each cone of every vertex u, we add an edge from u to the nearest neighbour v within that cone, with distance measured along the bisector of the cone (see Figure 1).

It is known that Θ_k-graphs are spanners (see Table 1 for a summary). In addition, various local routing algorithms with good routing ratios have been developed for Θ_k-graphs. A prime example is Θ*-routing*, also known as *cone-routing*. Θ-routing from s to t on a Θ_k-graph is defined as follows: Let u be the current vertex visited during the routing from s to t. If there exists a direct edge from u to t, follow this edge. Otherwise, follow the edge to the closest vertex in the cone of u containing t. This is repeated until the destination is reached.

[1] In this paper, we use the standard *real RAM* model of computation used in computational geometry. In this model, each *machine word* holds a real number or a $O(\log n)$-bit integer (such as a pointer or an index), and we assume that arithmetic operations (excluding the floor and ceiling functions for real numbers) on machine words take $O(1)$ time. Consequently, a $O(1)$-memory routing algorithm may store $O(\log n)$ bits.

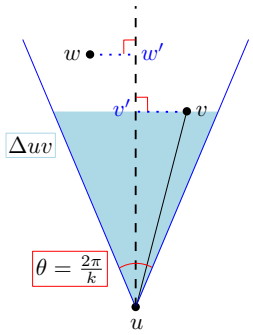


Figure 1 The creation of an edge (u, v) during the construction of a Θ_k-graph. The orthogonal projection v' of v is the closest to u in the cone.

Θ-routing is known to be competitive when used on Θ_k-graphs for $k \geq 7$. In addition to Θ-routing, various local routing algorithms with good routing ratios have been developed for Θ-graphs with smaller numbers of cones (see Table 1 for a summary).

Table 1 The current upper bounds on the spanning and routing ratio of Θ_k-graphs, with $\theta = \frac{2\pi}{k}$.

	Spanning Ratio	Routing Ratio
Θ_2 and Θ_3	Not spanners [12]	Not spanners [12]
Θ_4	17 [3]	17 [3]
Θ_5	$\frac{\sin(3\pi/10)}{\sin(2\pi/5)-\sin(3\pi/10)}$ [7]	-
Θ_6	2 [1]	2 [5]
$\Theta_{(4i+2)}$ for $i \geq 2$	$1 + 2\sin(\frac{\theta}{2})$ [4]	$\frac{1}{1-2\sin(\theta/2)}$ [16]
$\Theta_{(4i+3)}$ for $i \geq 1$	$\frac{\cos(\theta/4)}{\cos(\theta/2)-\sin(3\theta/4)}$ [4]	$1 + \frac{2\sin(\theta/2)\cdot\cos(\theta/4)}{\cos(\theta/2)-\sin(\theta/2)}$ [4]
$\Theta_{(4i+4)}$ for $i \geq 1$	$1 + \frac{2\sin(\theta/2)}{\cos(\theta/2)-\sin(\theta/2)}$ [4]	$1 + \frac{2\sin(\theta/2)}{\cos(\theta/2)-\sin(\theta/2)}$ [4]
$\Theta_{(4i+5)}$ for $i \geq 1$	$\frac{\cos(\theta/4)}{\cos(\theta/2)-\sin(3\theta/4)}$ [4]	$1 + \frac{2\sin(\theta/2)\cdot\cos(\theta/4)}{\cos(\theta/2)-\sin(\theta/2)}$ [4]

Unfortunately, the Θ-graph has a number of undesirable properties, including potentially a high maximum degree and a large diameter. The *ordered* Θ*-graph* alleviated these issues. It was introduced by Bose et al. [6] in 2004 to achieve logarithmic maximum degree and logarithmic diameter, in addition to the spanner property. An ordered Θ_k-graph is constructed by inserting the vertices one by one and connecting each vertex to the "closest" *previously* inserted vertex in each of the k cones.

However, achieving these additional properties comes at a price. For example, ordered Θ-graphs with 4, 5, or 6 cones are not spanners [8], despite the fact that their unordered counterparts are. In addition, ordered Θ-graphs have worse spanning ratios compared to Θ-graphs. The best known upper bounds and lower bounds on the spanning ratio of ordered Θ_k-graphs are summarized in Table 2.

Although we have good local routing algorithms for the Θ-graph, no local routing algorithm is known for the ordered Θ-graph. A major obstacle is that even if there are vertices in a cone, there may still not be any edges, due to the insertion order (see Figure 2). This makes the ordered Θ-graph much more challenging to route on than the Θ-graph – for example, Θ-routing fails on the ordered Θ-graph.

Table 2 The current upper and lower bounds on the spanning ratio of ordered Θ_k-graphs, with $\theta = \frac{2\pi}{k}$.

	Upper Bound	Lower Bound
Θ_3, Θ_4, Θ_5, and Θ_6	Not constant spanners [8]	Not constant spanners [8]
$\Theta_{(4i+2)}$ for $i \geq 2$	$\frac{1}{1-2\sin(\theta/2)}$ [6]	$\frac{1}{1-2\sin(\theta/2)}$ [8]
$\Theta_{(4i+3)}$ for $i \geq 1$	$\frac{1}{1-2\sin(\theta/2)}$ [6]	$\frac{\cos(\theta/4)+\sin(\theta)}{\cos(3\theta/4)}$ [8]
$\Theta_{(4i+4)}$ for $i \geq 1$	$1+\frac{2\sin(\theta/2)}{\cos(\theta/2)-\sin(\theta/2)}$ [8]	$1+\frac{2\sin(\theta/2)}{\cos(\theta/2)-\sin(\theta/2)}$ [8]
$\Theta_{(4i+5)}$ for $i \geq 1$	$\frac{1}{1-2\sin(\theta/2)}$ [6]	$1+\frac{2\sin(\theta/2)\cdot\cos(\theta/4)}{\cos(\theta/2)-\sin(3\theta/4)}$ [8]

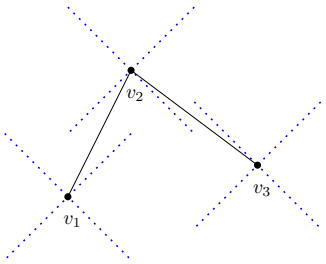


Figure 2 Three vertices v_1, v_2, and v_3 are inserted in the order v_1, v_2, v_3. When inserting v_2, an edge to v_1 is added. When inserting v_3, only an edge to v_2 is added, resulting in v_1 having vertex v_3 in a cone with no edges.

2 Technical overview

Our contribution is twofold: First, we show why so far no deterministic *memoryless h-local* routing algorithms have been found. Specifically, we prove that there does not exist a deterministic memoryless h-local routing algorithm on the ordered Θ_k-graph, for $h \geq 1$ and $k \geq 2$. We construct two ordered Θ_k-graphs L_h^k and R_h^k that cannot be distinguished in this routing model, which implies that any such algorithm fails on one of L_h^k and R_h^k. By doing so, we obtain the following theorem.

▶ **Theorem 1.** *There is no deterministic memoryless h-local routing algorithm (for any integer $h \geq 1$) capable of finding a path between every pair of source vertex s and destination vertex t on the ordered Θ_k-graph (for any integer $k \geq 2$).*

We then present a deterministic $O(1)$-memory 1-local routing algorithm $\mathcal{A}$ and prove that it converges to the destination in $O(n)$ hops (i.e., taking at most $O(n)$ edges). Our algorithm $\mathcal{A}(s,t)$ works in two phases: The first routes from the source s to the first inserted vertex v_1 by repeatedly moving to a neighbour with a smaller order. In the second phase, our algorithm routes from v_1 to the destination t by systematically exploring vertices that might lead to t. By exploring "candidate" vertices in a systematic way, we ensure that we do not need to keep track of the path. Instead, we can use a simple memoryless 1-local routing algorithm we call *ordered* Θ*-routing*, which allows us to implicitly keep track of the vertices we need to backtrack to by utilizing the geometric property of the ordered Θ-graph.

To prove that our algorithm $\mathcal{A}(s,t)$ converges, we consider the path P travelled by ordered Θ-routing from t to v_1 and show that our algorithm explores this path in the reverse direction. By arguing that any vertex is explored at most once and backtracked from at most once, we conclude that our algorithm reaches the destination in at most $O(n)$ hops and $O(kn)$ time. Consequently, we obtain the following theorem.

▶ **Theorem 2.** *Given an arbitrary source vertex $s \in V$ and the destination vertex $t \in V$ of an ordered Θ_k-graph $G = (V, E)$ (for an integer $k \geq 2$), the algorithm $\mathcal{A}(s, t)$ is a deterministic $O(1)$-memory 1-local routing algorithm which successfully routes from s to t in $O(n)$ hops. In particular, $\mathcal{A}(s, t)$ takes $O(kn)$ time in the worst-case.*

3 Preliminaries

Given a set of vertices $V \subset \mathbb{R}^2$ in the Euclidean plane, the ordered Θ_k-graph $G = (V, E)$ is a geometric network, with edges being undirected straight line segments, constructed as follows. For each vertex $u \in V$, we define $\rho(u)$ to be the order of insertion of u – i.e., if v_i is the i-th vertex inserted into G, for some positive integer $i \leq n$, where $n = |V|$, we let $\rho(v_i) = i$. We incrementally insert the vertices from the first to the last according to the pre-defined order ρ. Upon inserting each vertex $u \in V$, we partition the plane around u into k disjoint regions, by projecting k equally spaced rays from u, such that the angle between each pair of consecutive rays is $\theta = \frac{2\pi}{k}$ (see Figure 3(a)). We define a *cone* C^u of u to be a region in the plane between two consecutive rays originating from u. We orient the k cones such that the bisector of one of them coincides with the vertical half-line through u lying above u. Let the cone with such a bisector be C_0^u, from which we start numbering the cones in clockwise order around u. For the insertion of every other vertex $v \in V$, we orient and number the cones around v in the same way. We write C_i^v to denote the i-th cone of v, for $0 \leq i < k$. We say that v is the *apex* of C_i^v. We also define $C^v(w)$ to be the cone of v containing a vertex w.

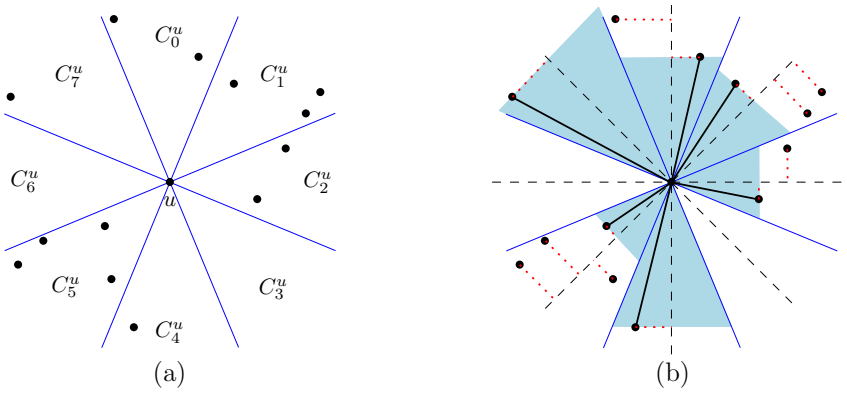


Figure 3 (a) the numbered cones around u and (b) the edges created in each of them.

Upon the insertion of each vertex $u \in V$, an edge is created to the "closest" *previously* inserted vertex in each cone C_i^u of u (see Figure 3(b)). More formally, an edge is created from u to v if (1) u is the apex of the cone that contains v such that $\rho(v) < \rho(u)$ and (2) for every vertex w within the cone such that $\rho(w) < \rho(u)$, $|uv'| \leq |uw'|$, where x' denotes the orthogonal projection of a vertex x onto the bisector of the cone of u that contains x. The canonical triangle Δuv corresponding to an edge created from u to v is defined by the boundary of the cone of u containing v and the line through v and its orthogonal projection onto the bisector. It is important to note that only the vertices inserted before u are considered for the edge creation process. This is why different orderings of vertex insertions result in different ordered Θ_k-graphs, even if the vertex set is the same.

For the insertion of each vertex $u \in V$, we store its order $\rho(u)$ on the vertex, in addition to the x- and y-coordinates of u. Since $1 \leq \rho(u) \leq n$ (where $n = |V|$), storing the order $\rho(u)$ requires at most $\lceil \log n \rceil$ extra bits per vertex, fitting in a single machine word.

It is important to note that a canonical triangle of the ordered Θ-graph is not necessarily empty, despite the fact that any canonical triangle in the Θ-graph is always empty of vertices in its interior. This is because the vertices are inserted incrementally and we can always insert a vertex in a canonical triangle between a pair of vertices previously inserted: Suppose, during the construction of an ordered Θ-graph, we have just inserted a vertex $v \in V$ with order $\rho(v) = k$. Let v' with order $\rho(v') < \rho(v) = k$ be the closest neighbour of a cone of v. At this stage, the canonical triangle $\Delta vv'$ is empty. However, we can insert the next vertex v'' with order $\rho(v'') = k + 1$ into the interior of $\Delta vv'$, making the canonical triangle non-empty.

For simplicity, in order to avoid tie-breaking for edge creation, we assume that V is in *general position*. Specifically, we will assume that, upon inserting each vertex during the construction of the ordered Θ-graph, it is clear which vertices lie in which cone (i.e., no previously inserted vertex lies on a boundary of a cone of the currently inserted vertex) and it is clear which vertex is the "closest" in each cone of a vertex (i.e., no vertex w lies on the boundary of the canonical triangle Δuv of the current vertex u and a neighboring vertex v, where $w \neq u$ and $w \neq v$). This assumption can be removed using standard techniques.

4 Local Routing on Ordered Θ_k-graphs

In this section, we will show that no deterministic *memoryless* local routing algorithm is guaranteed to reach the destination on the ordered Θ_k-graph, for any integer $k \geq 2$ (Section 4.1). As this motivates us to allow a small amount of memory, we will then present a deterministic $O(1)$-*memory* local routing algorithm that is guaranteed to reach the destination (Section 4.2).

We first define an important operation used by local routing algorithms.

▶ **Definition 3** (Hop). *We define a* hop *to be a single move or a forwarding operation from one vertex to its neighbour. Namely, taking a hop means we either (1) move from a vertex u to another vertex v through an edge, or (2) forward a message from u to v via an edge between them.*

4.1 Impossibility result for memoryless local routing

We show that no deterministic memoryless local routing algorithm works on the ordered Θ-graph. This impossibility result naturally motivates us to consider the next weakest class of local routing algorithms – deterministic $O(1)$-memory 1-local routing algorithm.

▶ **Theorem 1.** *There is no deterministic memoryless h-local routing algorithm (for any integer $h \geq 1$) capable of finding a path between every pair of source vertex s and destination vertex t on the ordered Θ_k-graph (for any integer $k \geq 2$).*

Proof. We present two ordered Θ_k-graphs such that any deterministic memoryless h-local routing algorithm A fails to route from s to t on one of these graphs. Each of these two graphs has $2h + 3$ vertices and $2h + 2$ edges, forming a simple path (see Figure 4).

Intuitively, we design two graphs, where one has a path from s to t to the left of s and t (L_h^k) and one to the right of s and t (R_h^k). By ensuring that these graphs are identical up to h hops from s, no h-local routing algorithm can tell them apart, and since it is deterministic and has no memory, it will always make the same choice at s, regardless of what path of the graph it has explored in the past.

Formally, we specify the location of the $2h + 3$ vertices of L_h^k as follows: $v_1 = (0, 0)$, $v_{2i} = (\frac{i}{h}, -i\epsilon)$ and $v_{2i+1} = (-\frac{i}{h}, -i\epsilon)$ for $i \in \{1, 2, \ldots, h\}$, $v_{2h+2} = (-2, -2h\epsilon)$, and $v_{2h+3} = (0, -\frac{2}{\tan(\theta/2)} - 3h\epsilon)$. Note that the y-coordinate of v_{2h+3} is given in terms of θ, ϵ, and h

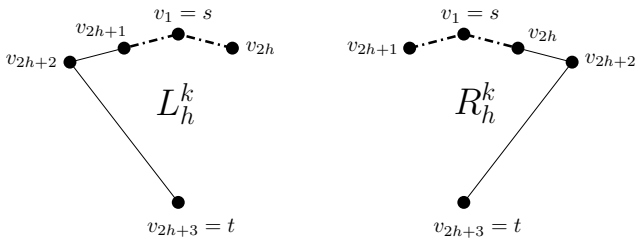


Figure 4 A pair of ordered Θ_k-graphs where any deterministic memoryless h-local routing algorithm fails on one of them.

such that all previously inserted vertices belong to $C_0^{v_{2h+3}}$. Here, ϵ is an arbitrarily small non-negative real number, such that for $j \in \{2, 3, \ldots, 2h+3\}$, a cone of v_j contains the vertices $v_{j'}$ for $j' \in \{1, 2 \ldots, j-1\}$. In particular, if no boundary of the cones of a vertex u coincides with the horizontal line through u, ϵ can simply be 0. We insert these vertices in ascending order: v_1 and then v_j for $j \in \{2, \ldots, 2h+3\}$. The creation of R_h^k is very similar to that of L_h^k. The only difference is that v_{2h+2} is placed at $(2, -2h\epsilon)$ in R_h^k.

We set s to be v_1 and t to be v_{2h+3} for both L_h^k and R_h^k, then we run A on both graphs. Consider the current vertex $u = s = v_1$ of routing. From the perspective of A, the local configuration at v_1 in L_h^k and the local configuration at v_1 in R_h^k are indistinguishable: The coordinate and ordering information of $u = v_1$, $t = v_{2h+3}$, and the h-neighbourhood of $u = v_1$ are identical between both graphs. Hence, any deterministic memoryless h-local routing algorithm A must make the same choice at v_1 both in L_h^k and R_h^k. If A decides to forward the message to v_2 from v_1, A can never reach v_3 in L_h^k, since A always deterministically makes the same choice at v_1 to forward the message to v_2 because the algorithm is memoryless, thereby going back and forth between v_1 and v_2 indefinitely. Similarly, if A decides to forward the message to v_3 from $s = v_1$, A cannot reach v_2 and thus t on R_h^k. ◀

4.2 $O(1)$-memory local routing

Now that we know we cannot route locally without using some memory, we shift to having a limited amount of it. In particular, our approach will store $O(1)$ information.

Our algorithm works in two phases, first to the lowest order vertex v_1 (Section 4.2.1) then to t (Section 4.2.2). The first phase repeatedly moves from the current vertex to a neighbour with a smaller order, which results in reaching v_1 from s without knowing v_1 in advance. The second phase routes from v_1 to t by systematically exploring vertices that might lead to t while implicitly keeping track of the vertices we need to backtrack to by utilizing the geometric property of the ordered Θ-graph.

4.2.1 Reaching vertex v_1

We consider a special vertex in the graph $G = (V, E)$ – the first inserted vertex $v_1 \in V$ with $\rho(v_1) = 1$. We can easily route from s to v_1 using the following algorithm that we call $\mathcal{A}^{down}$.

▶ **Algorithm 1** ($\mathcal{A}^{down}(s)$)**.** Given a source s, we initialize the current vertex u to be s. Then, we repeat the following process until we have $u = v_1$: Pick $v \in N(u)$ such that $\rho(v) < \rho(u)$ (if there are multiple options, pick an arbitrarily one among them, say the minimum) and update $u \leftarrow v$. Note that we do not know v_1 in advance before reaching it.

Below, we analyze the correctness and complexity of $\mathcal{A}^{down}$.

▶ **Lemma 4.** *Given an arbitrary source vertex $s \in V$ and the destination vertex being the first inserted vertex $v_1 \in V$ of an ordered Θ_k-graph $G = (V, E)$ (for an integer $k \geq 2$), the algorithm $\mathcal{A}^{down}(s)$ is a deterministic memoryless 1-local routing algorithm which successfully routes from s to v_1 in $O(n)$ hops. In particular, $\mathcal{A}^{down}(s)$ takes $O(kn)$ time in the worst-case.*

Now that we can route from s to v_1 using $\mathcal{A}^{down}(s)$, we move on to showing how to route from v_1 to t.

4.2.2 Finding the destination t

Starting from v_1, we will systematically explore vertices that might lead to t. Our strategy has two main components: exploration and backtracking. For the former we have to be able to tell which vertices have already been explored, while for the latter we have to know which vertex to backtrack to when the current vertex has no more valid neighbours to explore.

We first present the core component of the backtracking strategy: *ordered Θ-routing*. Given an ordered Θ-graph G and an arbitrary pair of source and destination vertices $s, t \in V$, we define *ordered Θ-routing* as follows.

▶ **Algorithm 2** (Ordered Θ-routing)**.** Initialize the current vertex $u \leftarrow s$. Then, we repeat this forwarding operation until reaching t or "getting stuck": If there exists an edge from u to t, take that edge – we are done. Otherwise, take the edge to the closest neighbour $v \in \Delta ut$ such that $\rho(v) < \rho(u)$. We call this single hop an *ordered Θ-routing step*. If both conditions are negative – i.e., (1) there does not exist an edge from u to t and (2) there does not exist an edge from u to a vertex $v \in \Delta ut$ with a smaller order – ordered Θ-routing is considered "stuck" and we terminate.

The ordered Θ-routing scheme we presented is inspired by the Θ-routing scheme developed for the regular Θ-graph. The main difference is that ordered Θ-routing considers the ordering information when determining which neighbour to forward the message to – it always chooses the closest neighbour with a smaller order.

Note that ordered Θ-routing from the source s to the destination t can only forward the message to the next vertex if either (1) there exists an edge from the current vertex u to the destination t or (2) there exists a vertex $v \in \Delta ut$ such that $\rho(v) < \rho(u)$. Thus, it can fail to reach the destination when neither condition is met. However, as we will see below, ordered Θ-routing is guaranteed to reach the destination if the destination is v_1, the first inserted vertex during the construction of the ordered Θ-graph.

▶ **Lemma 5.** *Given an arbitrary source $s \in V$ of any ordered Θ_k-graph $G = (V, E)$, ordered Θ-routing towards the first inserted vertex v_1 always reaches v_1.*

Proof. We prove the lemma by routing from s to v_1 using ordered Θ-routing (see Figure 5). We initialize the current vertex $u \leftarrow s$, and consider the two possible routing steps:

- If there is an edge from u to v_1, we can take that edge and reach v_1 – we are done.
- Otherwise, consider the canonical triangle Δuv_1. The fact that there does not exist an edge from u to v_1 implies that, when the current vertex u was inserted during the construction of the graph, there existed a vertex $v \in \Delta uv_1$ inserted before u (i.e., we have $\rho(u) > \rho(v)$), such that v was the closest neighbour in the cone $C^u(v_1)$. This means that there exists an edge from u to $v \in \Delta uv_1$, such that $\rho(u) > \rho(v)$. We take that edge. After moving from u to v, we update the current vertex u to be v.

Ordered Θ-routing from s to v_1 eventually converges. Since the order $\rho(v_1) = 1$ is the smallest among all vertices in V, the current vertex u has an edge either to v_1 or to a vertex $v \in \Delta uv_1$ such that $\rho(u) > \rho(v)$ by construction. So, the order $\rho(u)$ of the current vertex u

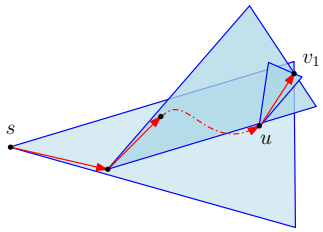


Figure 5 Ordered Θ-routing from s to v_1.

keeps decreasing by at least 1 for each step. Hence, within $n-1$ hops, the canonical triangle Δuv_1 becomes empty of vertices whose order is smaller than that of u, which implies there exists an edge from u to v_1. ◀

We can route from any vertex to v_1 using ordered Θ-routing. In particular, observe that we can replace $\mathcal{A}^{down}(s)$ with ordered Θ-routing from s to v_1 if every vertex stores v_1. However, to minimize space, we use $\mathcal{A}^{down}(s)$ instead.

In order to route from v_1 to t, we look for the reverse of the path traveled by ordered Θ-routing from t to v_1. To see which vertices are "worth exploring" to find such a path, we formally define the *exploration space* below. Conceptually, the exploration space builds a set of vertices by considering the ordered Θ-routing paths from all vertices to v_1, in order to cover the reverse of the ordered Θ-routing path from t to v_1.

▶ **Definition 6** (Exploration Space). *Given an ordered Θ-graph $G=(V,E)$, the first inserted vertex $v_1 \in V$, and an arbitrary destination vertex $t \in V$, the* exploration space $\mathcal{S}(v_1) \subseteq V$ *is a set of vertices defined recursively as follows:*

$$\mathcal{S}(v_1) := \{v_1\} \cup \bigcup_{v \in N^*(v_1)} \left(\{v\} \cup \bigcup_{v' \in N^*(v)} \left(\{v'\} \cup \bigcup_{v'' \in N^*(v')} (\cdots) \right) \right),$$

where we define the exploration candidate neighbours $N^*(u) \subseteq N(u)$ *of any vertex $u \in V$ to be the set of neighbours u' of u satisfying these two conditions:*

1. $\rho(u) < \rho(u') \leq \rho(t)$.
2. *Let $C^{u'}(v_1)$ be the cone of u' containing v_1 in its interior. Then, $C^{u'}(v_1)$ also contains u in its interior – i.e., we have $u \in C^{u'}(v_1)$.*

In particular, these two conditions are equivalent to saying that an ordered Θ-routing step from u' to v_1 leads to u (see Lemma 7 below for the proof).

Note that the *implication* of the two conditions (that an ordered Θ-routing step from u' to v_1 leads to u) is what we actually want to check, because we want to find the reverse of the path travelled by ordered Θ-routing from t to v_1. However, directly checking the implication would require 2-local information – given a current vertex u, accessing a neighbour $u' \in N(u)$ is 1-local, and accessing a neighbour $u'' \in N(u')$ of u', in order to check whether an ordered Θ-routing step from u' to v_1 leads to u'' such that $u''=u$, is 2-local. So, instead we check the two conditions (1) and (2), which is 1-local – we just have to compare the order of u, u', and t, and check the geometric statement $u \in C^{u'}(v_1)$, both of which are 1-local.

▶ **Lemma 7.** *Given an ordered Θ-graph $G=(V,E)$, the first inserted vertex $v_1 \in V$, an arbitrary destination vertex $t \in V$, an arbitrary current vertex $u \in V$, and a neighbour $u' \in N(u)$ of u, we have the following: $\rho(u) < \rho(u') \leq \rho(t)$ and $u \in C^{u'}(v_1)$ if and only if an ordered Θ-routing step from u' to v_1 leads to u and $\rho(u') \leq \rho(t)$.*

Proof. The fact that the ordered Θ-routing step and the inequality $\rho(u') \leq \rho(t)$ imply the two conditions follows by definition. Hence, we focus on the other proof direction. Suppose there exists $u' \in N(u)$ satisfying the first and second conditions. Then, $\rho(u') \leq \rho(t)$ trivially holds, and since $u' \in N(u)$, we have an edge $\{u, u'\} \in E$. The edge $\{u, u'\}$ must have been created upon the insertion of u' into G (when we inserted u into G, u' did not exist yet). This means that, when u' was inserted into G, u was the closest vertex in $C^{u'}(v_1)$. It follows that there does not exist a vertex $w \neq u$ such that w lies in the canonical triangle $\Delta u'u$ and the order of w is smaller than the order of u', because the existence of such vertex w contradicts the fact that u was the closest vertex in $C^{u'}(v_1)$ when u' was inserted into G. Note that we have $\Delta u'u \subset C^{u'}(v_1)$ by the second condition $u \in C^{u'}(v_1)$. Hence, an ordered Θ-routing step from u' to v_1 leads to u and $\rho(u') \leq \rho(t)$. ◀

It is important to ensure that the exploration space $\mathcal{S}(v_1)$ contains t, which we will argue through the lemma given below.

▶ **Lemma 8.** *Given an ordered Θ-graph $G = (V, E)$, the first inserted vertex $v_1 \in V$, an arbitrary destination vertex $t \in V$, the exploration space $\mathcal{S}(v_1)$ contains t.*

Consequently, our problem of locally routing from v_1 to t can be reduced to systematically exploring all vertices belonging to the exploration space $\mathcal{S}(v_1)$ defined above. Building upon the lemmas we have, we now present a deterministic $O(1)$-memory 1-local routing algorithm $\mathcal{A}^{up}$ that routes from v_1 to t, where $v_1 \in V$ is the first inserted vertex and $t \in V$ is an arbitrary destination vertex of the given ordered Θ_k-graph $G = (V, E)$ (for an integer $k \geq 2$). Conceptually, $\mathcal{A}^{up}$ mimics Depth-First Search (DFS) over the vertices in the exploration space $\mathcal{S}(v_1)$. However, unlike how DFS keeps track of the current exploration path from the source by storing a sequence of vertices in a stack so that backtracking can be done by popping the top vertex from the stack, $\mathcal{A}^{up}$ does not explicitly store the exploration path – our local routing algorithm delegates it to the geometric property of the ordered Θ-graph, as backtracking can be done with an ordered Θ-routing step. Below is the full description.

▶ **Algorithm 3** ($\mathcal{A}^{up}(v_1, t)$)**.** Do the following:

1. Initialize the previous vertex u^p to be *null*.
2. Initialize the current vertex u to be v_1.
3. Initialize a bit named *state* to be 1, indicating whether the previous hop was exploration or backtracking.
4. Repeat this block until $u = t$:
   - If $state = 1$ (the previous hop was exploration):
     Let l be the horizontal half-line originating from u towards the right. Sweep l in counter-clockwise direction (see Figure 6). Let $v \in N(u)$ be the first neighbour that touches l. Check whether $v \in N^*(u)$ by checking:
     a. $\rho(u) < \rho(v) \leq \rho(t)$.
     b. $u \in C^v(v_1)$.

     If indeed $v \in N^*(u)$, then we explore v from u by updating the current vertex u, i.e., we set $u \leftarrow v$. Otherwise, we keep sweeping l in counter-clockwise direction, checking whether each $v \in N(u)$ is an exploration candidate neighbour in $N^*(u)$ – if one is found, explore that exploration candidate neighbour, updating u. If l reaches its original position, we perform an ordered Θ-routing step from u to v_1 to backtrack to a vertex u' (observe that $u \in N^*(u')$), updating the current vertex $u \leftarrow u'$, the previous vertex $u^p \leftarrow u$, and the state bit $state \leftarrow 0$.

- Else, i.e., $state = 0$ (the previous hop was backtracking):
  Let l be the half-line originating from u through u^p. Sweep l in counter-clockwise direction as usual, starting from this position, until it becomes the half-line pointing to the right. For each $v \in N(u)$ that touches l, check whether it is an exploration candidate neighbour by checking the two conditions in the same way as above. If an exploration candidate neighbour is found, explore it and update the variables accordingly, setting u to be the neighbour and *state* to be 1. If l becomes the half-line pointing to the right, we perform an ordered Θ-routing step from u to v_1 to backtrack to a vertex u', updating the variables as in the previous case, setting $u \leftarrow u'$, $u^p \leftarrow u$, and $state \leftarrow 0$.

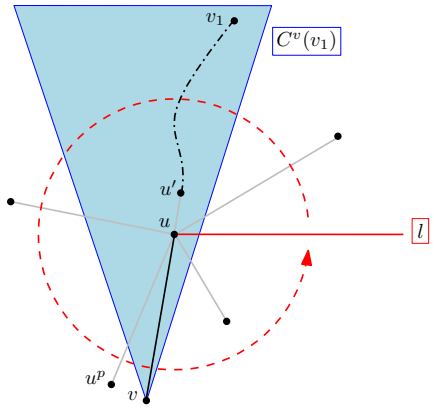


Figure 6 Illustration of $\mathcal{A}^{up}(v_1, t)$.

Before analyzing the correctness of $\mathcal{A}^{up}$, we will first argue that $\mathcal{A}^{up}$ is a deterministic $O(1)$-memory 1-local routing algorithm: There is no randomness involved in the algorithm, so it is deterministic. It only stores v_1, u^p, and the *state* bit. The routing algorithm does not have to store the current vertex u because we are at said vertex, which means we can trivially access the current node without storing it. Hence, the algorithm only uses $O(1)$-memory. The algorithm only accesses the 1-neighbourhood $N(u)$ of u, so it is a 1-local routing algorithm. Therefore, we have the following lemma.

▶ **Lemma 9.** *$\mathcal{A}^{up}$ is a deterministic $O(1)$-memory 1-local routing algorithm.*

Next, we prove the convergence of $\mathcal{A}^{up}$, by showing that the algorithm, starting from v_1, reaches t in $O(n)$ hops. To deduce this, we will prove three lemmas.

▶ **Lemma 10.** *Let u be the current vertex of $\mathcal{A}^{up}(v_1, t)$. Then, $u \in \mathcal{S}(v_1)$.*

▶ **Lemma 11.** *Any vertex $v \in \mathcal{S}(v_1)$ is explored by $\mathcal{A}^{up}(v_1, t)$ at most once.*

Proof. We prove the lemma by induction.

The base case is when $v = v_1$. At the beginning, the algorithm is at v_1. Clearly, v_1 cannot be explored again from a neighbour $v \in N(v_1)$, since the first condition $(\rho(v) < \rho(v_1) \leq \rho(t))$ for v_1 being in $N^*(v)$ cannot be met, because $\rho(v_1) = 1$ is the minimum among all vertices of the graph. Hence, v_1 is explored at most once.

The inductive case is when $v \neq v_1$. Suppose v was explored for the first time from u. Our inductive hypothesis is that u was explored at most once. We will prove by contradiction that v cannot be explored again: For the sake of contradiction, suppose $\mathcal{A}^{up}(v_1, t)$ explored

the vertex v for the first time from u and then for a second time from u' (see Figure 7). Then, we have $v \in N^*(u)$ and $v \in N^*(u')$, since the algorithm explicitly checks whether v is an exploration candidate neighbour of u and u'. Firstly, we will show that $u \neq u'$. Having $u = u'$ means that we visited v from u twice. By induction hypothesis, u was explored at most once. So, to visit v twice from u, after exploring v from u for the first time, we have to backtrack from v to u through an ordered Θ-routing step and then explore v again from u. However, this is impossible since the counter-clockwise sweep of l, which determines the exploration order of the vertices in $N^*(u)$, is done only once. Hence, we cannot explore $v \in N^*(u)$ twice from u, meaning that $u \neq u'$.

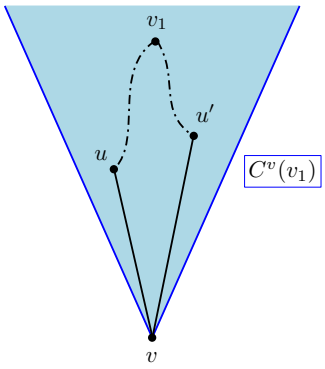


Figure 7 Illustration of a situation where v is explored from u and from u', which leads to a contradiction.

Consider $C^v(v_1)$. Since we explored v from u and then from u', both u and u' lie in the cone $C^v(v_1)$ by the second condition of Definition 6. Also, both u and u' are inserted before v by the first condition of Definition 6. By the general position assumption, only one of u or u' is the closest to v in $C^v(v_1)$. Consequently, during the construction of the ordered Θ-graph, an edge is created from v to only one of u or u', meaning that we have either $v \notin N(u)$ or $v \notin N(u')$ – i.e., v is not a neighbour of either u or u'. This contradicts to the fact that v was explored from u and then from u'. Hence, $v \neq v_1$ can be explored at most once.

Therefore, any vertex $v \in \mathcal{S}(v_1)$ is explored by $\mathcal{A}^{up}(v_1, t)$ at most once. ◀

Observe that, since the algorithm is guaranteed to visit any vertex at most once, we do not have to keep track of which vertices have been explored and which have not.

▶ **Lemma 12.** *For any vertex $v \in \mathcal{S}(v_1)$ explored by $\mathcal{A}^{up}(v_1, t)$, an ordered Θ-routing step is performed at v towards v_1 at most once – i.e, the algorithm backtracks from v at most once.*

By Lemma 10, the local routing algorithm $\mathcal{A}^{up}(v_1, t)$ only explores vertices that belong to $\mathcal{S}(v_1)$. By Lemma 11 and Lemma 12, there can be at most $O(n)$ exploration and backtracking hops because $\mathcal{S}(v_1) \subseteq V$. In addition, we know that $t \in \mathcal{S}(v_1)$ by Lemma 8. Therefore, $\mathcal{A}^{up}(v_1, t)$ requires $O(n)$ hops overall to reach t, implying the following lemma.

▶ **Lemma 13.** *$\mathcal{A}^{up}(v_1, t)$ reaches t in $O(n)$ hops.*

Next, we show that $\mathcal{A}^{up}(v_1, t)$ performs a linear number of operations to determine all exploration candidate neighbours.

▶ **Lemma 14.** *Given any current vertex u during the local routing procedure of $\mathcal{A}^{up}(v_1, t)$, let a* candidate verification *of $v \in N(u)$ be the operations required for checking the two conditions required for having $v \in N^*(u)$. Then, $\mathcal{A}^{up}(v_1, t)$ performs at most $O(kn)$ candidate verifications before reaching t, where k is the number of cones of the ordered Θ-graph.*

4.2.3 Routing from s to t

As $\mathcal{A}^{down}$ routes from s to v_1 and $\mathcal{A}^{up}$ routes from v_1 to t, we obtain our full algorithm.

▶ **Algorithm 4** ($\mathcal{A}(s,t)$)**.** Given the source vertex $s \in V$ and the destination $t \in V$, the algorithm routes from s to t by performing the following two steps:

1. Execute $\mathcal{A}^{down}(s)$ to route from s to v_1.
2. Execute $\mathcal{A}^{up}(v_1, t)$ to route from v_1 to t.

Consequently, we obtain the following theorem.

▶ **Theorem 2.** *Given an arbitrary source vertex $s \in V$ and the destination vertex $t \in V$ of an ordered Θ_k-graph $G = (V, E)$ (for an integer $k \geq 2$), the algorithm $\mathcal{A}(s,t)$ is a deterministic $O(1)$-memory 1-local routing algorithm which successfully routes from s to t in $O(n)$ hops. In particular, $\mathcal{A}(s,t)$ takes $O(kn)$ time in the worst-case.*

5 Conclusion

In this paper, we addressed the following problem with a positive result: *Does there exist a local routing algorithm that works on the ordered Θ-graph?* We first proved there does not exist a deterministic memoryless h-local routing on the ordered Θ_k-graph for any integers $h \geq 1$ and $k \geq 2$. Then, we presented a deterministic $O(1)$-memory local routing algorithm that is guaranteed to reach the destination in $O(n)$ hops and $O(kn)$ time on the ordered Θ_k-graph for any $k \geq 2$. To the best of our knowledge, the algorithm we provided is the first local routing strategy that is successful on the ordered Θ_k-graph.

Although our local routing algorithm $\mathcal{A}(s,t)$ can route between any source and destination, we remark that the total routing path length can be arbitrarily high, which means our local routing algorithm is not c-competitive.

Therefore, one natural open question is whether there exists a c-competitive local routing algorithm using a small amount of memory on the ordered Θ-graph. Things that make finding such an algorithm difficult include the fact that ordered Θ-graphs are generally not plane, the proofs used to bound the spanning ratio by Bose et al. [8] and Bose et al. [6] are highly non-local, and the commonly used properties (such as empty canonical triangles) do not hold for these graphs.

References

1 N. Bonichon, C. Gavoille, N. Hanusse, and D. Ilcinkas. Connections between theta-graphs, Delaunay triangulations, and orthogonal surfaces. In *Proceedings of the 36th International Workshop on Graph Theoretic Concepts in Computer Science*, pages 266–278. Springer Berlin Heidelberg, 2010. doi:10.1007/978-3-642-16926-7_25.

2 P. Bose, J.-L. De Carufel, and O. Devillers. Expected complexity of routing in θ_6 and half-θ_6 graphs. *Journal of Computational Geometry*, 11(1):212–234, 2020. doi:10.20382/jocg.v11i1a9.

3 P. Bose, J.-L. De Carufel, D. Hill, and M. Smid. On the spanning and routing ratio of the directed theta-four graph. *Discrete & Computational Geometry*, 71(3):872–892, 2024. doi:10.1007/s00454-023-00597-8.

4 P. Bose, J.-L. De Carufel, P. Morin, A. van Renssen, and S. Verdonschot. Towards tight bounds on theta-graphs: More is not always better. *Theoretical Computer Science*, 616:70–93, 2016. doi:10.1016/j.tcs.2015.12.017.

5 P. Bose, R. Fagerberg, A. van Renssen, and S. Verdonschot. Optimal local routing on Delaunay triangulations defined by empty equilateral triangles. *SIAM Journal on Computing*, 44(6):1626–1649, 2015. doi:10.1137/140988103.

6 P. Bose, J. Gudmundsson, and P. Morin. Ordered theta graphs. *Computational Geometry*, 28(1):11–18, 2004. doi:10.1016/j.comgeo.2004.01.003.

7 P. Bose, D. Hill, and A. Ooms. Improved spanning ratio of the theta-5 graph. *Journal of Computational Geometry*, 15(1):66–87, 2024. doi:10.20382/jocg.v15i1a3.

8 P. Bose, P. Morin, and A. van Renssen. The price of order. *International Journal of Computational Geometry & Applications*, 26(03n04):135–149, 2016. doi:10.1142/S0218195916600013.

9 P. Bose and M. Smid. On plane geometric spanners: A survey and open problems. *Computational Geometry*, 46(7):818–830, 2013. doi:10.1016/j.comgeo.2013.04.002.

10 K. Clarkson. Approximation algorithms for shortest path motion planning. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pages 56–65, 1987. doi:10.1145/28395.28402.

11 E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959. doi:10.1007/BF01386390.

12 N. M. El Molla. Yao spanners for wireless ad hoc networks. Master's thesis, Villanova University, 2009. URL: http://www.csc.villanova.edu/~mdamian/StudentTheses/NawarThesis.pdf.

13 J. M. Keil. Approximating the complete Euclidean graph. In *Proceedings of the 1st Scandinavian Workshop on Algorithm Theory*, pages 208–213, 1988. doi:10.1007/3-540-19487-8_23.

14 E. F. Moore. The shortest path through a maze. In *Proceedings of the International Symposium on the Theory of Switching*, pages 285–292, 1959.

15 G. Narasimhan and M. Smid. *Geometric spanner networks*. Cambridge University Press, 2007. doi:10.1017/CBO9780511546884.

16 J. Ruppert and R. Seidel. Approximating the d-dimensional complete Euclidean graph. In *Proceedings of the 3rd Canadian Conference on Computational Geometry*, pages 207–210, 1991. URL: https://cccg.ca/proceedings/1991/paper50.pdf.

A Parameterized Study of Secluded Structures in Directed Graphs

Jonas Schmidt
Bocconi University, Milan, Italy

Shaily Verma
Hasso Plattner Institute, Potsdam, Germany

Nadym Mallek
Hasso Plattner Institute, Potsdam, Germany

Abstract

Given an undirected graph G and an integer k, the Secluded Π-Subgraph problem asks you to find a maximum size induced subgraph that satisfies a property Π and has at most k neighbors in the rest of the graph. This problem has been extensively studied; however, there is no prior study of the problem in directed graphs. This question has been mentioned by Jansen et al. [ISAAC'23].

In this paper, we initiate the study of Secluded Subgraph problems in directed graphs by incorporating different notions of neighborhoods: in-neighborhood, out-neighborhood, and their union. Formally, we call these problems {In, Out, Total}-Secluded Π-Subgraph, where given a directed graph G and an integer k, we want to find an induced subgraph satisfying Π of maximum size that has at most k in/out/total-neighbors in the rest of the graph, respectively. We investigate the parameterized complexity of these problems for different properties Π. In particular, we prove the following parameterized results:

- We design an FPT algorithm for the Total-Secluded Strongly Connected Subgraph problem when parameterized by k.
- We show that the Out-Secluded $\mathcal{F}$-Free Subgraph problem with parameter k is W[1]-hard, where $\mathcal{F}$ is a family of directed graphs except any subgraph of a star graph whose edges are directed towards the center. This result also implies that In/Out-Secluded DAG is W[1]-hard, unlike the undirected variants of the two problems, which are FPT.
- We design an FPT-algorithm for In/Out/Total-Secluded α-Bounded Subgraph when parameterized by k, where α-bounded graphs are a superclass of tournaments.
- For undirected graphs, we improve the best-known FPT algorithm for Secluded Clique by providing a faster FPT algorithm that runs in time $1.6181^k n^{\mathcal{O}(1)}$.

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

Keywords and phrases Secluded Subgraph, Parametrized Complexity, Directed Graphs, Strong Connectivity

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.53

Related Version *Full Version*: https://arxiv.org/abs/2502.06048 [21]

1 Introduction

Finding substructures in graphs that satisfy specific properties is a ubiquitous problem. This general class of problems covers many classical graph problems such as finding maximum cliques, Steiner trees, or even shortest paths. Another compelling property to look for in a substructure is its isolation from the remaining graph. This motivated Chechik et al. [1], to introduce the concept of *secluded subgraphs*. Formally, in the Secluded Π-Subgraph problem, given an undirected graph G, the goal is to find a maximum size subset of vertices $S \subseteq \mathrm{V}(G)$ such that the subgraph induced on S fulfills a property Π and has a neighborhood $|\mathrm{N}(S)| \leq k$ where k is a natural number.

© Jonas Schmidt, Shaily Verma, and Nadym Mallek;
licensed under Creative Commons License CC-BY 4.0

Table 1 Our main results for directed and undirected problems. WC stands for weakly connected. All FPT results are with respect to parameter k and all hardness results are with respect to parameter $k + w$. For the entry marked with $*$, the undirected algorithm from [15] immediately generalizes to the total-secluded setting (for finite $\mathcal{F}$). The problems marked with (?) are open.

Property Π	**In- / Out-Secluded**		**Total-Secluded**	
WC $\mathcal{F}$-Free Subgraph	W[1]-hard	Thm. 2	FPT*	[15]
WC DAG	W[1]-hard	Cor. 3	?	
α-Bounded Subgraph	FPT	Thm. 4	FPT	Thm. 5
Strongly Connected Subgraph	?		FPT	Thm. 1
Clique	FPT in time $1.6181^k n^{\mathcal{O}(1)}$			Thm. 6

This problem has been studied extensively for various properties Π such as paths [23, 17, 8, 1], Steiner trees [1, 8], induced trees [7, 11], subgraphs free of forbidden induced subgraphs [11, 15], and more [22]. Most of these studies focus on the parameterized setting, due to the strong relation to vertex deletion and separator problems, which are foundational in parameterized complexity. Our problem fits in that category since the neighborhood can be considered a $(S, V \setminus S)$-separator.

While the undirected Secluded Π-Subgraph problem has been explored and widely understood in prior work [15, 11, 7], the directed variant has not yet been studied, although it is a natural generalization and was mentioned as an interesting direction by Jansen et al. [15]. Directed graphs naturally model real-world systems with asymmetric interactions, such as social networks with unidirectional follow mechanisms or information flow in communication systems. Furthermore, problems such as Directed Feedback Vertex Set, Directed Multicut, and Directed Multiway Cut underline how directedness can make a fascinating and insightful difference when it comes to parameterized complexity [2, 13, 19, 4]. These problems and their results provide a ground for studying directed secluded subgraph problems, motivating the need to investigate them systematically.

In this paper, we introduce three natural directed variants of the Secluded Π-Subgraph problem, namely Out-Secluded Π-Subgraph, In-Secluded Π-Subgraph, and Total-Secluded Π-Subgraph. These problems limit either the out-neighborhood of S, its in-neighborhood, or the union of both. The out/in-neighborhood of a set S is the set of vertices in $\mathrm{V}(G) \setminus S$ reachable from S via an outgoing/incoming edge. The problems corresponding to different types of neighborhoods can be encountered in real-life networks. In privacy-aware social network analysis, one might aim to identify a community with minimal external exposure. Similarly, robust substructures with limited connectivity to vulnerable components are critical in cybersecurity. These real-world motivations further emphasize the need to explore and formalize directed variants of the Secluded Π-Subgraph problem.

For any property Π, we formulate our general problem as follows. Notice that in-secluded and out-secluded are equivalent for all properties that are invariant under the transposition of the edges. For this reason, we mostly focus on total-neighborhood and out-neighborhood.

X-Secluded Π-Subgraph $(\mathrm{X} \in \{\mathrm{In}, \mathrm{Out}, \mathrm{Total}\})$

Parameter: An integer $k \in \mathbb{N}$
Input: A directed graph G with vertex weights $\omega \colon V \to \mathbb{N}$ and an integer $w \in \mathbb{N}$
Output: An k-X-secluded set $S \subseteq \mathrm{V}(G)$ of weight $\omega(S) \geq w$ that satisfies Π, or report that none exists.

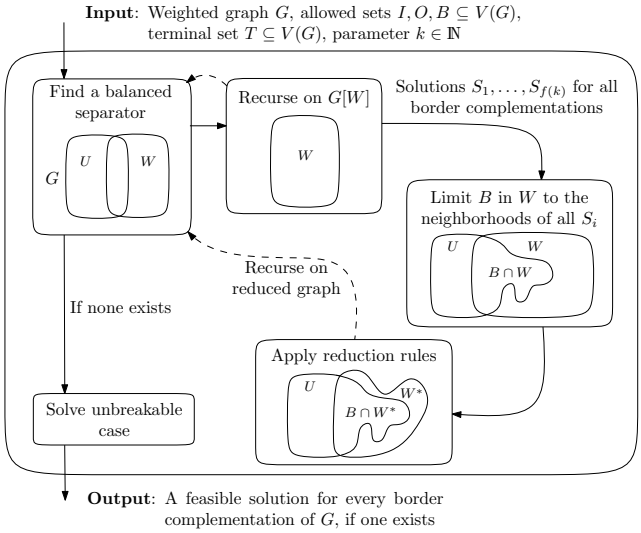


Figure 1 An illustration of the general recursive understanding algorithm used in Section 2. There are two recursive calls in total, highlighted with dashed arrows. As defined later, only vertices inside B are allowed to be in the neighborhood of a solution. W is chosen to be the side of the separation with a smaller intersection with T.

Our Contribution

In this subsection, we present the results we obtained for the problem. See Table 1 for an overview of the results we obtained in this paper.

Strongly Connected Subgraph. We show that the Total-Secluded Strongly Connected Subgraph problem is fixed-parameter tractable when parameterized with k. Precisely, we prove the following result:

▶ **Theorem 1.** *Total-Secluded Strongly Connected Subgraph is solvable in time* $2^{2^{2^{\mathcal{O}(k^2)}}} n^{\mathcal{O}(1)}$.

We design our FPT algorithm for Total-Secluded Strongly Connected Subgraph problem using recursive understanding, a technique introduced by [12] and recently used successfully for various parameterized problems [3, 11, 6, 16]. Specifically, [16] prove a meta-result stating that if a problem is FPT on highly-connected graphs and expressible in *Counting Monadic Second-Order Logic*, it is also FPT on general graphs. Thereby, this theorem allows to shortcut the analysis and algorithm for the breakable case of recursive understanding. However, it comes at the price of being nonconstructive and not giving a concrete bound on the runtime. For this reason, it is still valuable to apply recursive understanding directly. In future work, it could be promising to apply and generalize this theorem for directed graphs. We visualize the overall structure of the algorithm in Figure 1.

On a high level, recursive understanding algorithms work by first finding a small balanced separator of the underlying undirected graph. If no suitable balanced separator exists, the graph must be highly connected, which makes the problem simpler to solve. In the other case, we reduce and simplify one side of the separator while making sure to keep an equivalent set of solutions in the whole graph. By choosing parameters in the right way, this process reduces one side of the separator enough to invalidate the balance of the separator. Therefore, we have made progress and can iterate with another balanced separator or reach the base case.

In our case, looking for a separator of size at most k makes the framework applicable. Crucially, this is because in any secluded subgraph $G[S]$, where $S \subseteq \mathrm{V}(G)$, the neighborhood $\mathrm{N}(S)$ acts as a separator between S and $\mathrm{V}(G) \setminus \mathrm{N}[S]$. Therefore, if no balanced separator of size at most k exists, we can deduce that either S itself or $\mathrm{V}(G) \setminus S$ must have a small size. This observation makes the problem significantly easier to solve in this case, using the color coding technique developed in [3].

In the other case, we can separate our graph into two balanced parts, U and W, with a separator P of size $|P| \leq k$. Now, our goal is to solve the same problem recursively for one of the sides, say W and replace that side with an equivalent graph of bounded size that only contains all necessary solutions. However, finding subsets of solutions is not the same as finding solutions; the solution S for the whole graph could heavily depend on including some vertices in U. That being said, the different options for this influence are limited. At most 2^k different subsets X of P could be part of the solution. For any such X, the solution can only interact across P in a limited number of ways. For finding strongly connected subgraphs, we have to consider for which pair $(x_1, x_2) \in X \times X$ there already is a x_1-x_2-path in the U-part of the complete solution S. This allows us to construct a new instance for every such possibility by encoding X and the existing paths into W. These instances are called *boundary complementations*. We visualize the idea of this construction in Figure 2.

Fundamentally, we prove that an optimal solution for the original graph exists, that coincides in W with an optimal solution to the boundary complementation graph in which U is replaced. Hence, we restrict the space of solutions to only those whose neighborhood in W coincides with the neighborhood of an optimal solution to some boundary complementation. The restricted instance consists of a bounded-size set B of vertices that could be part of $\mathrm{N}(S)$ and components in $W \setminus B$ that can only be included in S completely or not at all. We introduce graph *extensions* to formalize when exactly these components play the same role in a strongly connected subgraph. Equivalent extensions can then be merged and compressed into equivalent extensions of bounded size. In total, this guarantees that W is shrunk enough to invalidate the previous balanced separator, and we can restart the process.

$\mathcal{F}$-Free Subgraph. For any family of graphs $\mathcal{F}$, a graph G is called $\mathcal{F}$-Free if it does not contain any graph in $\mathcal{F}$ as an induced subgraph. Depending on the context both $\mathcal{F}$ and G are either both undirected or both directed. The widely-studied Secluded $\mathcal{F}$-Free Subgraph problem on undirected graphs is FPT (with parameter k) using recursive understanding [11] or branching on important separators [15] when we restrict it to connected solutions. We study the directed version of the problem and surprisingly, it turns out to be W[1]-hard for almost all forbidden graph families $\mathcal{F}$ even with respect to the parameter $k + w$. Precisely, we prove the following theorem.

▶ **Theorem 2.** *Let $\mathcal{F}$ be a non-empty set of directed graphs such that no $F \in \mathcal{F}$ is a subgraph of an inward star. Then, Out-Secluded $\mathcal{F}$-Free Weakly Connected Subgraph is W[1]-hard with respect to the parameter $k + w$ for unit weights.*

We establish an almost complete dichotomy that highlights the few cases of families $\mathcal{F}$ for which the problem remains tractable. One of these exceptions is if $\mathcal{F}$ contains an edgeless graph of any size, where we employ our algorithm for Out-Secluded α-Bounded Subgraph. Theorem 2 also implies the following result for the directed variant of Secluded Tree problem.

▶ **Corollary 3.** *Out-Secluded Weakly Connected DAG is W[1]-hard with parameter $k + w$ for unit weights.*

α-Bounded Subgraph & Clique. In the undirected setting, the Secluded Clique problem is natural and has been studied specifically. There is an FPT-algorithm running in time $2^{\mathcal{O}(k \log k)} n^{\mathcal{O}(1)}$ through contracting twins [11]. The previous best algorithm however uses the general result for finding secluded $\mathcal{F}$-free subgraphs in time $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ [15]. By using important separators, they require time at least $4^k n^{\mathcal{O}(1)}$. This property is naturally generalizable to directed graphs via tournament graphs. We go one step further.

The independence number of an undirected or directed graph G is the size of the maximum independent set in G (or its underlying undirected graph). If G has independence number at most α, we also call it *α-bounded*. This concept has been used to leverage parameterized results from the simpler tournament graphs to the larger graph class of α-bounded graphs [20, 10, 18]. We prove the following results:

▶ **Theorem 4.** *Out-Secluded α-Bounded Subgraph is solvable in time $(2\alpha+2)^k n^{\alpha+\mathcal{O}(1)}$.*

▶ **Theorem 5.** *Total-Secluded α-Bounded Subgraph is solvable in time $(\alpha+1)^k n^{\alpha+\mathcal{O}(1)}$.*

We achieve the goal via a branching algorithm, solving Secluded α-Bounded Subgraph for all neighborhood definitions in FPT time (Theorems 4 and 5). Our algorithm initially picks a vertex subset $U \subseteq \mathrm{V}(G)$ and looks only for solutions in the two-hop neighborhood of U. A structural property of α-bounded graphs guarantees that any optimal solution is found in this way. On a high level, the remaining algorithm depends on two branching strategies. First, we branch on forbidden structures in the two-hop neighborhood of U, to ensure that it becomes α-bounded. Second, we branch on farther away vertices to reach a secluded set.

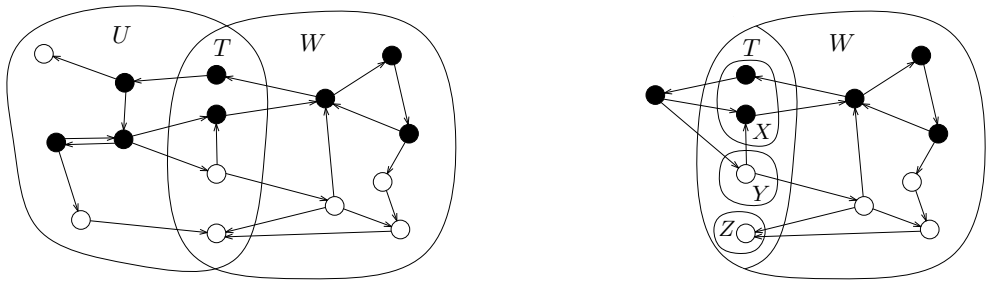
Note that Theorem 5 is inherently also an undirected result. Furthermore, the ideas behind the algorithm can in turn be used for the simpler undirected Secluded Clique problem. By a closer analysis of these two high-level rules, we arrive at a branching vector of $(1, 2)$ for Secluded Clique. This results in the following runtime, a drastic improvement on the previous barrier of $4^k n^{\mathcal{O}(1)}$.

▶ **Theorem 6.** *Secluded Clique is solvable in time $1.6181^k n^{\mathcal{O}(1)}$.*

Organization. We consider Total-Secluded Strongly Connected Subgraph and prove Theorem 1 in Section 2. The hardness result about Out-Secluded $\mathcal{F}$ Weakly Connected Subgraph in Theorem 2 is proved in Section 3. In Section 4, we give the algorithm for Out-Secluded α-Bounded Subgraph and prove Theorem 4. Due to space constraints, the algorithms and proofs of Theorems 5 and 6 can be found in the full version [21]. Proofs of statements marked with $(\star)$ are also deferred to the full version.

Notation. Let G be a directed graph. For a vertex $v \in \mathrm{V}(G)$, we denote the *out-neighborhood* by $\mathrm{N}^+(v) = \{u \mid (v,u) \in \mathrm{E}(G)\}$ and the *in-neighborhood* by $\mathrm{N}^-(v) = \{u \mid (u,v) \in \mathrm{E}(G)\}$. The *total-neighborhood* is defined as $\mathrm{N}(v) = \mathrm{N}^+(v) \cup \mathrm{N}^-(v)$. We use the same notation for sets of vertices $S \subseteq \mathrm{V}(G)$ as $\mathrm{N}^+(S) = \bigcup_{v \in S} \mathrm{N}^+(v) \setminus S$. Furthermore, for all definitions, we also consider their *closed* version that includes the vertex or vertex set itself, denoted by $\mathrm{N}^+[v] = \mathrm{N}^+(v) \cup \{v\}$. For a vertex set $S \subseteq \mathrm{V}(G)$, we write $G[S]$ for the subgraph induced by S or $G - S$ for the subgraph induced by $\mathrm{V}(G) \setminus S$. We also use $G - v$ instead of $G - \{v\}$.

When we refer to a component of a directed graph, we mean a component of the underlying undirected graph, that is, a maximal set that induces a weakly connected subgraph. In contrast, a strongly connected component refers to a maximal set that induces a strongly connected subgraph. For standard parameterized definitions, we refer to [5].



(a) A strongly connected subgraph S of a graph G with $k = 5$ neighbors. Black vertices are part of S.

(b) The boundary complementation that admits an equivalent feasible solution when setting $k' := k-2$.

Figure 2 A visualization of a solution in the original graph and a solution in a boundary complementation. Every partial solution in U can be represented by a boundary complementation.

2 Total-Secluded Strongly Connected Subgraph

In this section, we investigate the Total-Secluded Strongly Connected Subgraph problem, or TSSCS for short. First, we prove that the problem is NP-hard in general graphs, motivating analysis of its parameterized complexity.

▶ **Theorem 7** ($\star$). *TSSCS is NP-hard, even for unit weights.*

The proof of Theorem 7 also shows that TSSCS is W[1]-hard when parameterized by w, since w in the proof also only depends on the parameter for Clique. In the following subsections, we describe the recursive understanding algorithm to solve TSSCS parameterized by k. We follow the framework by [3, 11] and first introduce generalized problems in Section 2.1. In Section 2.2, we solve the case of unbreakable graphs. We introduce graph *extensions* in Section 2.3 as a framework to formulate our reduction rules and full algorithm in Section 2.4.

2.1 Boundaries and boundary complementations

In this subsection, we first define an additional optimization problem that is useful for recursion. Then, we describe a problem-specific *boundary complementation*. Finally, we define the auxiliary problem that our algorithm solves, which includes solving many similar instances from the optimization problem.

Max TSSCS

Input: A directed graph G, subsets $I, O, B \subseteq \mathrm{V}(G)$, a weight function $\omega\colon \mathrm{V}(G) \to \mathbb{N}$, and an integer $k \in \mathbb{N}$
Output: A set $S \subseteq \mathrm{V}(G)$ that maximizes $\omega(S)$ subject to $I \subseteq S$, $O \cap S = \emptyset$, $N(S) \subseteq B$, $|\mathrm{N}(S)| \leq k$, and $G[S]$ strongly connected, or report that no feasible solution exists.

Note that this problem generalizes the optimization variant of TSSCS by setting $I := O := \emptyset$ and $B := \mathrm{V}(G)$. However, Max TSSCS allows us to put more constraints on recursive calls, enforcing vertices to be included or excluded from the solution and neighborhood.

▶ **Definition 8** (Boundary Complementation). *Let $\mathcal{I} = (G, I, O, B, \omega, k)$ be a* Max TSSCS *instance. Let $T \subseteq \mathrm{V}(G)$ be a set of* boundary terminals *with a partition $X, Y, Z \subseteq T$ and let $R \subseteq X \times X$ be a relation on X. Then, we call the instance $(G', I', O', B, \omega', k')$ a* boundary complementation *of $\mathcal{I}$ and T if*

1. *G' is obtained from G by adding vertices $u_{(a,b)}$ for every $(a, b) \in R$ and edges $(a, u_{(a,b)})$, $(u_{(a,b)}, b)$, and for every $y \in Y$ additionally $(u_{(a,b)}, y)$,*
2. *$I' \coloneqq I \cup X \cup \{u_{(a,b)} \mid (a, b) \in R\}$,*
3. *$O' \coloneqq O \cup Y \cup Z$,*
4. *$\omega'(v) \coloneqq \omega(v)$ for $v \in \mathrm{V}(G)$ and $\omega'(u_{(a,b)}) \coloneqq 0$ for $(a, b) \in R$, and*
5. *$k' \leq k$.*

See Figure 2 for an example boundary complementation. The intuition here should be that if we take the union of G with any other graph H and only connect H to G at the vertices in T, then (X, Y, Z, R) encodes all possibilities of how a solution in $G \cup H$ could behave from G's point of view. So, for any solution S to Max TSSCS in $G \cup H$, there is some boundary complementation for G in which we can solve and exchange $S \cap G$ for that solution. Later, we prove a statement that is similar to this intuition.

To employ recursive understanding, we need a boundaried version of the problem. Intuitively, this problem is the same as the previous Max TSSCS but for a small part of the graph we want to try out every possibility, giving many very similar instances. This small part will later represent a separator to a different part of the graph.

Boundaried Max TSSCS

Input: A Max TSSCS instance $\mathcal{I} = (G, I, O, B, \omega, k)$ and a set of boundary terminals $T \subseteq \mathrm{V}(G)$ with $|T| \leq 2k$
Output: A solution to Max TSSCS for each boundary complementation $\mathcal{I}'$ of $\mathcal{I}$ and T, or report that no solution exists.

To even have a chance to solve this problem, we need to make sure that there are not too many boundary complementations. The following lemma bounds that number in terms of k.

▶ **Lemma 9** ($\star$). *For a Max TSSCS instance (G, I, O, B, ω, k) and $T \subseteq \mathrm{V}(G)$, there are at most $3^{|T|}2^{|T|^2}(k+1)$ many boundary complementations, which can be enumerated in time $2^{\mathcal{O}(|T|^2)}n^{\mathcal{O}(1)}$.*

2.2 Unbreakable Case

This subsection gives the algorithm for the base case of our final recursive algorithm, when no balanced separator exists. We start by giving the definitions of separations and unbreakability.

▶ **Definition 10** (Separation). *Given two sets $A, B \subseteq \mathrm{V}(G)$ with $A \cup B = \mathrm{V}(G)$, we say that (A, B) is a* separation of order $|A \cap B|$ *if there is no edge with one endpoint in $A \setminus B$ and the other endpoint in $B \setminus A$.*

▶ **Definition 11** (Unbreakability). *Let $q, k \in \mathbb{N}$. An undirected graph G is* (q, k)-unbreakable *if for every separation (A, B) of G of order at most k, we have $|A \setminus B| \leq q$ or $|B \setminus A| \leq q$.*

Due to space constraints, we defer any further details on the algorithm for the unbreakable case to the full version [21]. It uses standard color coding techniques from [3, 11].

▶ **Theorem 12** ($\star$). *Boundaried Max TSSCS on (q, k)-unbreakable graphs can be solved in time $2^{\mathcal{O}(k^2 \log(q))}n^{\mathcal{O}(1)}$.*

2.3 Compressing Graph Extensions

Before we give the complete algorithm, we define a routine that compresses a part of the graph. We aim for two crucial properties in the compressed part. First, the part after compression should be equivalent to the part before compression in terms of which strongly connected components can be formed. Second, we want the size of the compressed part to be functionally bounded by the size of remaining graph.

To achieve this goal, we first formally define sufficient properties to reason about this equivalence and bound the number of equivalence classes. First, we define the notion of a graph *extension*, a way to extend one graph with another. The extension will play the role of the compressed part of the graph. This concept allows us to speak more directly about graph properties before and after exchanging a part of the graph with a different one.

▶ **Definition 13** (Extension). *Given a directed graph G, we call a pair (D, E_{GD}) an* extension *of G if D is a directed graph and $E_{GD} \subseteq (\mathrm{V}(G) \times \mathrm{V}(D)) \cup (\mathrm{V}(D) \times \mathrm{V}(G))$ is a set of pairs between G and D. We name the graph $Ext_G(D, E_{GD}) \coloneqq (\mathrm{V}(G) \cup \mathrm{V}(D), \mathrm{E}(G) \cup \mathrm{E}(D) \cup E_{GD})$, that can be created from the extension, G* extended by *(D, E_{GD}).*

We use extensions to construct extended graphs. Intuitively, an extension of G is a second graph D together with an instruction E_{GD} on how to connect D to G.

Next, we identify three important attributes of extensions in our context. Later, we show that these give a sufficient condition on when two extensions form the same strongly connected subgraphs. For this, consider a directed graph G with an extension (D, E_{GD}). For $U \subseteq \mathrm{V}(D)$, we write $\mathrm{N}^-_{E_{GD}}(U)$ as a shorthand for $\mathrm{N}^-_{\mathrm{Ext}_G(D,E_{GD})}(U)$, that is, all $v \in \mathrm{V}(G)$ with $(v, u) \in E_{GD}$ for some $u \in U$. Define $\mathrm{N}^+_{E_{GD}}(v)$ analogously. Write $\mathrm{SCC}(D)$ for the *condensation* of D, where every strongly connected component C of D is contracted into a single vertex. Define $\mathcal{S}(D, E_{GD}), \mathcal{T}(D, E_{GD}) \subseteq 2^{\mathrm{V}(G)}$ such that

$$\mathcal{S}(D, E_{GD}) \coloneqq \left\{ \mathrm{N}^-_{E_{GD}}(U) \;\middle|\; U \subseteq \mathrm{V}(D) \text{ is a source component in } \mathrm{SCC}(D) \right\} \text{ and}$$
$$\mathcal{T}(D, E_{GD}) \coloneqq \left\{ \mathrm{N}^+_{E_{GD}}(U) \;\middle|\; U \subseteq \mathrm{V}(D) \text{ is a sink component in } \mathrm{SCC}(D) \right\},$$

that is, for every strongly connected source component C in $\mathrm{SCC}(D)$, $\mathcal{S}(D, E_{GD})$ contains the set of all $v \in \mathrm{V}(G)$ such that $(v, u) \in E_{GD}$ for some $u \in C$ and analogously for $\mathcal{T}(D, E_{GD})$. Furthermore, define

$$\mathcal{C}(D, E_{GD}) \coloneqq \left\{ (a, b) \in \mathrm{V}(G)^2 \;\middle|\; \text{there is a } d_1\text{-}d_2\text{-path in } D \text{ with } (a, d_1), (d_2, b) \in E_{GD} \right\},$$

that is, all (a, b) such that there is an a-b-path in $\mathrm{Ext}_G(D, E_{GD})$, whose intermediate vertices and edges belongs to D. Refer to Figure 3 for examples of extensions and the three sets.

▶ **Definition 14** (Equivalent Extensions). *Let G be a directed graph. We say that two extensions (D_1, E_{GD_1}) and (D_2, E_{GD_2}) of G are* equivalent *if*

$$(\mathcal{S}(D_1, E_{GD_1}), \mathcal{T}(D_1, E_{GD_1}), \mathcal{C}(D_1, E_{GD_1})) = (\mathcal{S}(D_2, E_{GD_2}), \mathcal{T}(D_2, E_{GD_2}), \mathcal{C}(D_2, E_{GD_2})).$$

Clearly, extension equivalence defines an equivalence relation. The next statement reveals the motivation behind the definition of extension equivalence. It gives us a sufficient condition for two extensions being exchangeable in a strongly connected subgraph.

▶ **Lemma 15.** *Let G be a directed graph with two equivalent extensions (D_1, E_{GD_1}) and (D_2, E_{GD_2}). Let $U \subseteq \mathrm{V}(G)$ be nonempty such that the extended graph $Ext_{G[U]}(D_1, E_{G[U]D_1})$ is strongly connected. Then $Ext_{G[U]}(D_2, E_{G[U]D_2})$ is also strongly connected.*

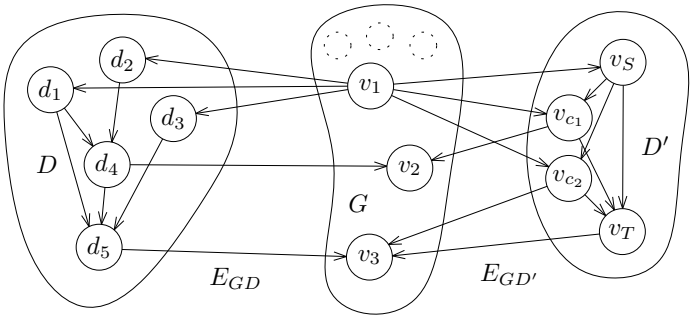


Figure 3 Two example extensions of a graph G. Observe that $\mathcal{S}(D, E_{GD}) = \{\{v_1\}\}$, $\mathcal{T}(D, E_{GD}) = \{\{v_3\}\}$, and $\mathcal{C}(D, E_{GD}) = \{(v_1, v_2), (v_1, v_3)\}$. The extension $(D', E_{GD'})$ not only has the same sets $\mathcal{S}, \mathcal{T}, \mathcal{C}$ and is thereby equivalent; it is also the compressed extension of (D, E_{GD}). Since all sources d_1, d_2, d_3 have the same in-neighborhood, they are represented by the single vertex v_S.

Proof. We construct a v_1-v_2-path for all $v_1, v_2 \in U \cup \mathrm{V}(D_2)$ that only uses edges in $\mathrm{E}(D_2)$, E_{GD_2}, and $G[U]$ by case distinction.

Paths $U \to U$. Let $u_1, u_2 \in U$. If there is a path from u_1 to u_2 in $G[U]$, this path also exists after exchanging (D_1, E_{GD_1}) to (D_2, E_{GD_2}). If the path passes through D_1, since $\mathcal{C}(D_1, E_{GD_1}) = \mathcal{C}(D_2, E_{GD_2})$, we can exchange all subpaths through D_1 by subpaths through D_2.

Paths $\mathrm{V}(D_2) \to U$. Let $v \in \mathrm{V}(D_2), u \in U$. We construct a v-u-path by first walking from v to any sink component T in $\mathrm{SCC}(D_2)$. If there is no edge $(t, u') \in E_{GD_2}$ with $t \in T, u' \in U$ that we can append, since $\mathcal{T}(D_1, (D_1, E_{GD_1})) = \mathcal{T}(D_2, (D_2, E_{GD_2}))$, there must also be a sink component in $\mathrm{SCC}(D_1)$ with no outgoing edge to U. However, this is a contradiction to the fact that $\mathrm{Ext}_{G[U]}(D_1, E_{G[U]D_1})$ is strongly connected with nonempty U. Therefore, we can find a (t, u') to append for some $t \in T, u' \in U$. From u', there is already a path to u, as proven in the first case.

Paths $U \to \mathrm{V}(D_2)$. Next, we construct a u-v-path backwards by walking from v backwards to a source s in D_2. Analogously, there is an edge $(u', s) \in E_{GD_2}$ for some $u' \in U$ since $\mathcal{S}(D_1, E_{GD_1}) = \mathcal{S}(D_2, E_{GD_2})$, which we append. From u, there is a path to u', as proven in the first case, which we prepend to the rest of the path.

Paths $\mathrm{V}(D_2) \to \mathrm{V}(D_2)$. Let $v_1, v_2 \in \mathrm{V}(D_2)$. To construct a v_1-v_2-path, we can just walk from v_1 to any $u \in U$ and from there to v_2 as shown before. ◀

Furthermore, observe that the union of two extensions creates another extension where source, sink and connection sets correspond exactly to the union of the previous sets. Hence, the union of two equivalent extensions will again be equivalent. This fact is formalized in the next observation and will turn out useful in later reduction rules.

▶ **Observation 16.** *Let G be a directed graph with two equivalent extensions (D_1, E_{GD_1}) and (D_2, E_{GD_2}). Consider the extension defined by $D \coloneqq (\mathrm{V}(D_1) \cup \mathrm{V}(D_2), \mathrm{E}(D_1) \cup \mathrm{E}(D_2))$ and $E_{GD} \coloneqq E_{GD_1} \cup E_{GD_2}$. Then (D, E_{GD}) is equivalent to (D_1, E_{GD_1}) and (D_2, E_{GD_2}).*

Now, we finally define our compression routine, which compresses an extension to a bounded size equivalent extension. If an extension is strongly connected, it is easy to convince yourself that it is always possible to compress the extension to a single vertex. Otherwise, we add one source vertex per neighborhood set in $\mathcal{S}(D, E_{GD})$ as well as one sink vertex per neighborhood set in $\mathcal{T}(D, E_{GD})$, realizing the same $\mathcal{S}$ and $\mathcal{T}$. Then, we add

vertices in between suitable source and sink vertices to realize exactly the same connections in $\mathcal{C}$ without creating additional ones. The result of a compression is visualized in Figure 3. Now, we describe the procedure formally.

Let G be a directed graph with an extension (D, E_{GD}). Our *compression routine* returns an extension that we call *compressed extension*, denoted as $\mathrm{Comp}_G(D, E_{GD})$.

Compression Routine.

- If D is strongly connected, we contract D to one vertex v and remove self-loops and multiple edges. We adjust E_{GD} by using v instead of $\mathrm{V}(D)$ and removing multiple edges.
- Otherwise, $\mathrm{Comp}_G(D, E_{GD}) \coloneqq (D', E_{GD'})$, where $(D', E_{GD'})$ is an extension such that

$$\begin{aligned}\mathrm{V}(D') &\coloneqq \{v_S \mid S \in \mathcal{S}(D, E_{GD})\} \cup \{v_T \mid T \in \mathcal{T}(D, E_{GD})\} \cup \{v_c \mid c \in \mathcal{C}(D, E_{GD})\},\\ E_{GD'} &\coloneqq \{(s, v_S) \mid S \in \mathcal{S}(D, E_{GD}), s \in S\} \cup \{(v_T, t) \mid T \in \mathcal{T}(D, E_{GD}), t \in T\}\\ &\quad \cup \{(a, v_c), (v_c, b) \mid c = (a, b) \in \mathcal{C}(D, E_{GD})\}.\end{aligned}$$

  To define $\mathrm{E}(D)$, consider every source component C_s and sink component C_t in $\mathrm{SCC}(D)$ such that C_t is reachable from C_s in D. Let $S \coloneqq \mathrm{N}^-_{E_{GD}}(C_s)$ and $T \coloneqq \mathrm{N}^-_{E_{GD}}(C_t)$ be the corresponding sets in $\mathcal{S}(D, E_{GD})$ and $\mathcal{T}(D, E_{GD})$.
  - Add the edge (v_S, v_T) to $\mathrm{E}(D)$.
  - For every $c = (a, b) \in \mathcal{C}(D, E_{GD})$ that satisfies $(s, b) \in \mathcal{C}(D, E_{GD})$ and $(a, t) \in \mathcal{C}(D, E_{GD})$ for every $s \in S, t \in T$, add the edges (v_S, v_c) and (v_c, v_T) to $\mathrm{E}(D)$.

Now we go on to prove the properties that are maintained while compressing. Then, we bound the size of a compressed extension and thus also the number of equivalence classes.

▶ **Lemma 17.** *Let G be a directed graph with an extension (D, E_{GD}) and let $(D', E_{GD'})$ be the compressed extension of (D, E_{GD}). Then, the following are true.*

1. *If D is weakly connected, then D' is also weakly connected.*
2. *D is strongly connected if and only if D' is strongly connected.*
3. *$(D', E_{GD'})$ is equivalent to (D, E_{GD}).*

Proof. For the first property, assume that D is weakly connected. We know by definition that every sink in D' is reached by at least one source. Consider a v_c with $c = (a, b) \in \mathcal{C}(D, E_{GD})$. To show that v_c is connected to some v_S and v_T, consider the path from d_1 to d_2 in D that realizes this connection. There must be a source component C_S and a sink component C_T in $\mathrm{SCC}(D)$ such that d_1 is reachable from C_S and C_T is reachable from d_2. Therefore, any vertex in C_S can also reach d_2 and d_1 can reach every vertex in C_T. By definition of compression, these two components ensure that v_c is connected.

It remains to show that any source is reachable by any other source in the underlying undirected graph. Let $v_S, v_{S'}$ be two sources in D' with corresponding source components C, C' in $\mathrm{SCC}(D)$. Since D is weakly connected, there is a path from C to C' in the underlying undirected graph. Whenever the undirected path uses an edge in a different direction than the one before, we extend the path to first keep using edges in the same direction until a source or sink component is reached and then go back to the switching point. This new path can directly be transferred to D', where we only keep the vertices corresponding to source and sink components. By definition, this is still a path in D' that connects v_S to $v_{S'}$, and D' is weakly connected

The second property is simple to verify, since strongly connected graphs are by definition compressed to single vertices. If D is not strongly connected, D' will have at least one source and one sink that are not the same.

Regarding the equivalence, we create one source for every $S \in \mathcal{S}(D, E_{GD})$ with the same set of incoming neighbors and create no other sources. Therefore, $\mathcal{S}(D', E_{GD'}) = \mathcal{S}(D, E_{GD})$ and $\mathcal{T}(D', E_{GD'}) = \mathcal{T}(D, E_{GD})$ follows analogously. For every connection $c \in \mathcal{C}(D, E_{GD})$, we create v_c in D' that realizes this connection. Therefore, we know that $\mathcal{C}(D', E_{GD'}) \supseteq \mathcal{C}(D, E_{GD})$. Since v_c is only reachable from sources and reaches only sinks that do not give new connections, we arrive at $\mathcal{C}(D', E_{GD'}) = \mathcal{C}(D, E_{GD})$. ◀

We also bound the number of possible different compression outputs as well as their size.

▶ **Lemma 18** (⋆). *For a directed graph G, there can be at most $2^{2 \cdot 2^{|\mathrm{V}(G)|} + |\mathrm{V}(G)|^2}$ different compressed extensions. Furthermore, every compressed extension has at most $2^{|\mathrm{V}(G)|+1} + |\mathrm{V}(G)|^2$ vertices.*

In the past section, we have defined graph extensions and an equivalence relation on them that captures the role they can play in forming strongly connected subgraphs. We have presented a way to compress an extension such that its size only depends on the size of G, while remaining equivalent. In the next section, we will use this theory to design reduction rules for Boundaried Max TSSCS. Crucially, we view components outside of the set B as extensions, which allows us to keep only one compressed extension of each equivalence class.

2.4 Solving Boundaried Max TSSCS

We start by giving some reduction rules for a Boundaried Max TSSCS instance $\mathcal{I} = (G, I, O, B, \omega, k, T)$. Additionally, we assume that $T \subseteq B$ to ensure that we do not change T when changing $G - B$. This condition will always be satisfied in our algorithm. A (⋆) after a reduction rule denotes that its proof of safeness can be found in the full version [21].

The first reduction rule extends the sets I and O to whole components of $G - B$. This is possible since no solution can include only part of a component without its neighborhood intersecting the component. Remember that components always refer to weakly connected components.

▶ **Reduction Rule 19** (⋆). *Let Q be a component of $G - B$. If $Q \cap O \neq \emptyset$, set $O = O \cup \mathrm{N}[Q]$. If $\mathrm{N}[Q] \cap I \neq \emptyset$, set $I = I \cup Q$. If both cases apply, the instance has no solution.*

If this reduction rule is no longer applicable, every component in $G - B$ is either completely in I, completely in O, or intersects with none of the two.

From this point, we will use extensions from the previous section for components of $G - B$. Namely, for a component Q of $G - B$, let E_{BQ} be all the edges with exactly one endpoint in B and one in Q. Then, $(G[Q], E_{BQ})$ defines an extension of $G - Q$. For simplicity, we also refer to this extension as (Q, E_{BQ}). Hence, we also use $\mathcal{S}$, $\mathcal{T}$, and $\mathcal{C}$, as well as Comp_{G-Q} for these extensions.

The next reduction rule identifies a condition under which a component Q can never be part of a solution, namely, if Q includes strongly connected components with no in-neighbors or no out-neighbors, which is exactly the case if the empty set is in $\mathcal{S}(Q, E_{BQ})$ or $\mathcal{T}(Q, E_{BQ})$.

▶ **Reduction Rule 20** (⋆). *Let Q be a component of $G - B$ such that $G[Q]$ is not strongly connected. If $\emptyset \in \mathcal{S}(Q, E_{BQ}) \cup \mathcal{T}(Q, E_{BQ})$, include Q into O.*

Note that after Reduction Rules 19 and 20 have been applied exhaustively, every source in Q has incoming edges from B, and every sink in Q has outgoing edges to B. Finally, we have one more simple rule, which removes vertices $v \in O \setminus B$. It relies on the fact that by Reduction Rule 19, we also have $\mathrm{N}(v) \subseteq O$.

▶ **Reduction Rule 21.** *If Reduction Rule 19 is not applicable, remove $O \setminus B$ from G.*

The previous reduction rules were useful to remove trivial cases and extend I and O. From now on, we assume that the instance is exhaustively reduced by Reduction Rules 19–21. Therefore, any component of $G - B$ is either contained in I or does not intersect I and O.

The next two rules will be twin type reduction rules that allow us to bound the number of remaining components. If there are two components of $G - B$ that form equivalent extensions it is enough to keep one of them, since they fulfill the same role in forming a strongly connected subgraph. The reduction rules rely on Lemma 15 and Observation 16 to show that equivalent extensions can replace each other and can be added to any solution.

▶ **Reduction Rule 22.** *Let Q_1, Q_2 be components of $G - B$ such that both $G[Q_1]$ and $G[Q_2]$ are not strongly connected and (Q_1, E_{BQ_1}) and (Q_2, E_{BQ_2}) are equivalent. Delete Q_2 and increase the weight of some $q \in Q_1$ by $\omega(Q_2)$. If $Q_2 \cap I \neq \emptyset$, set $I = I \cup Q_1$.*

Proof of Safeness. By the previous reduction rules, components of $G - B$ can only be included as a whole or not at all. Notice that since $\mathcal{C}(Q_1, E_{BQ_1}) = \mathcal{C}(Q_2, E_{BQ_2})$ and Reduction Rule 20, we get $\mathrm{N}(Q_1) = \mathrm{N}(Q_2)$. Let S be a solution to the old instance. We differentiate some cases.

If $S \cap (Q_1 \cup Q_2) = \emptyset$, we know that also $\mathrm{N}(S) \cap (Q_1 \cup Q_2) = \emptyset$. Therefore, the neighborhood size and strong connectivity of S do not change in the new instance, and it is also a solution. If S includes only one of Q_1 and Q_2, assume without loss of generality $S \cap (Q_1 \cup Q_2) = Q_1$, since Q_1 is not strongly connected by itself, the solution must include vertices of B. Because $\mathrm{N}(Q_1) = \mathrm{N}(Q_2)$, the solution must also include Q_2, a contradiction. If S includes both Q_1 and Q_2, we claim that $S' \coloneqq S \setminus Q_2$ is a solution for the new instance. The neighborhood size and weight clearly remain unchanged. Strong connectivity follows by Observation 16 and Lemma 15. The last two cases also show that if a solution had to include at least one of Q_1 and Q_2, that is, $(Q_1 \cup Q_2) \cap I \neq \emptyset$, any solution for the reduced instance must also include Q_1. Therefore, the adaptation to I is correct.

Let S be a solution to the reduced instance. Again, if S does not include vertices from Q_1, then S will immediately be a solution to the old instance. If $Q_1 \subseteq S$, then $S' \coloneqq S \cup Q_2$ will be a solution to the old instance by Observation 16 and Lemma 15. ◀

For strongly connected components, the rule is different, since we have to acknowledge the fact that they can be a solution by themselves. For every such component we destroy strong connectivity of smaller-weight equivalent component, which can then be reduced by Reduction Rule 22.

▶ **Reduction Rule 23.** *Let Q_1, Q_2 be components of $G - B$ that are also strongly connected with (Q_1, E_{BQ_1}) and (Q_2, E_{BQ_2}) equivalent and $\omega(Q_1) \geq \omega(Q_2)$. Add a vertex q_2' with edges (q_2, q_2') and (q_2', v), for all $q_2 \in Q_2$ and $v \in \mathrm{N}(Q_2)$. Set $\omega(q_2') = 0$.*

Proof of Safeness. Let S be a solution of the old instance. If $S \cap (Q_1 \cup Q_2) = \emptyset$, then S is also a solution for the new instance. If S includes only one of Q_1 and Q_2, then we must have $S \in \{Q_1, Q_2\}$, so $S' \coloneqq Q_1$ is a solution for the new instance with $\omega(S') \geq \omega(S)$. If S includes both Q_1 and Q_2, adding q_2' to S obviously gives a solution of the same weight.

For a solution of the reduced instance S, we can simply remove q_2' if it is inside for a solution to the old instance. ◀

Using both Reduction Rules 22 and 23 exhaustively makes sure that there are at most two components per extension equivalence class left. The last rule compresses the remaining components to equivalent components of bounded size.

Algorithm 1 The recursive understanding algorithm for Boundaried Max TSSCS.

def `Solve`$(G,\ I,\ O,\ B,\ \omega,\ k,\ T)$**:**
  $q \leftarrow 2^{2^{2^{ck^2}}}$ for a suitable constant c
  if G *is* $((2q+1)q2^k, k)$*-unbreakable* **then**
    return solve the problem using Theorem 12
  else
    $(U, W) \leftarrow (q,k)$-separation of G with $|T \cap W \setminus U| \leq k$
    $(\tilde{G}, \tilde{I}, \tilde{O}, \tilde{B}, \tilde{\omega}, k, \tilde{T}) \leftarrow$ restriction to W with $\tilde{T} = (T \cap W) \cup (U \cap W)$
    $\mathcal{R} \leftarrow$ `Solve`$(\tilde{G},\ \tilde{I},\ \tilde{O},\ \tilde{B},\ \tilde{\omega},\ k,\ \tilde{T})$
    $\mathcal{N} \leftarrow \tilde{T} \cup \bigcup_{R \in \mathcal{R}} \mathrm{N}(R) \cap W$
    $\hat{B} \leftarrow (B \cap U) \cup (B \cap \mathcal{N})$
    $(G^*, I^*, O^*, \hat{B}^*, \omega^*, k^*, T^*) \leftarrow$ reduce $(G, I, O, \hat{B}, \omega, k, T)$ with Lemma 25
    return `Solve`$(G^*,\ I^*,\ O^*,\ \hat{B}^*,\ \omega^*,\ k^*,\ T^*)$
  end

▶ **Reduction Rule 24.** *Let Q be a component of $G - B$ that is not equal to its compressed extension. Replace Q by its equivalent compressed extension and set the weight such that $\omega(Q') = \omega(Q)$. If $Q \cap I \neq \emptyset$, set $I = I \cup Q'$.*

Proof of Safeness. Since Q is not strongly connected, it can only be part of a solution that includes some vertices from $G - Q$. Thus, we can apply Lemmas 15 and 17, and the old instance and the new instance have exactly the same strongly connected subgraphs. Because of $\mathcal{C}(Q, E_{BQ}) = \mathcal{C}(Q', E_{BQ'})$ and Reduction Rule 20, we get $\mathrm{N}(Q) = \mathrm{N}(Q')$. Since $\omega(Q') = \omega(Q)$, the rule is safe. ◀

This finally allows us to bound the size of $G - B$. In the next lemma, we summarize the progress of our reduction rules and apply the bounds from the previous section. Note that we need the stronger bound using the neighborhood of the components instead of simply B.

▶ **Lemma 25** (⋆). *Let $\mathcal{Q}$ be a set of components of $G - B$ with total neighborhood size $h := \left|\bigcup_{Q \in \mathcal{Q}} \mathrm{N}(Q)\right|$. Then we can reduce the instance, or there are at most $2^{2^{h+1}+h^2}\left(2^{h+1} + h^2\right) = 2^{\mathcal{O}(2^h)}$ vertices in $\mathcal{Q}$ in total.*

Executing the reduction rules in the proposed order guarantees termination after $\mathcal{O}(n)$ applications. The total execution takes at most $n^{\mathcal{O}(1)}$ time.

▶ **Lemma 26** ([11, Lemma 3])**.** *Given an undirected graph G, there is an algorithm with runtime $2^{\mathcal{O}(\min\{q,k\}\log(q+k))}n^{\mathcal{O}(1)}$ that either finds a (q,k)-separation of G or correctly reports that G is $((2q+1)q2^k, k)$-unbreakable.*

Now, we have all that it takes to solve our intermediate problem. The main idea of the algorithm is to shrink B to a bounded size, by solving the problem recursively. Once B is bounded, we apply our reduction rules by viewing components of $G - B$ as extensions, removing redundant equivalent extensions and compressing them. Thereby, we also bound the size and number of components of $G - B$ in terms of $|B|$ using Lemma 25. By choosing suitable constants, we can show that this decreases the total size of G, which will make progress to finally reduce it to the unbreakable case.

▶ **Theorem 27** (⋆). *Boundaried Max TSSCS can be solved in time $2^{2^{2^{\mathcal{O}(k^2)}}}n^{\mathcal{O}(1)}$.*

Proof. Let $\mathcal{I} = (G, I, O, B, \omega, k, T)$ be our Boundaried Max TSSCS instance. See Algorithm 1 for a more compact description of the algorithm. A high level display of the approach can be found in Figure 1. Define $q = 2^{2^{2^{ck^2}}}$ for a constant c. We later show that a suitable c must exist.

First, we run the algorithm from Lemma 26 on the underlying undirected graph of G with q and k. If it is $((2q+1)q2^k, k)$-unbreakable, we solve the instance directly using the algorithm from Theorem 12.

Therefore, assume that we have a (q,k)-separation (U, W). Without loss of generality, since $|T| \leq 2k$ we can assume that $|T \cap W \setminus U| \leq k$. Thus, we can construct a new instance to solve *the easier side* of the separation. Take $\tilde{G} = G[W], \tilde{I} = I \cap W, \tilde{O} = O \cap W, \tilde{B} = B \cap W$, write $\tilde{\omega}$ for the restriction of ω to W, and set $\tilde{T} = (T \cap W) \cup (U \cap W)$. Since $|U \cap W| \leq k$, also $\left|\tilde{T}\right| \leq 2k$ holds and $\tilde{I} \coloneqq (\tilde{G}, \tilde{I}, \tilde{O}, \tilde{B}, \tilde{\omega}, k, \tilde{T})$ is a valid instance, which we solve recursively.

Let $\mathcal{R}$ be the set of solutions found in the recursive call. For $R \in \mathcal{R}$, define $N_R = \mathrm{N}(R) \cap W$. Define $\mathcal{N} = \tilde{T} \cup \bigcup_{R \in \mathcal{R}} N_R$.

We now restrict B in W to use only vertices in the neighborhood that have been neighbors in a solution in $\mathcal{R}$, that is, only vertices in $\mathcal{N}$. Define $\hat{B} = (B \cap U) \cup (B \cap \mathcal{N})$. We now replace B in our instance with $\hat{B}$ and apply all our reduction rules exhaustively to arrive at the instance $(G^*, I^*, O^*, \hat{B}^*, \omega^*, k^*, T^*)$. Finally, we also solve this instance recursively and return the solutions after undoing the reduction rules.

Correctness. We already proved that the reduction rules and the algorithm for the unbreakable case are correct. The main statement we have to show is that we can replace B with $\hat{B}$ without throwing away important solutions. That means that the instances $(G, I, O, B, \omega, k, T)$ and $\hat{\mathcal{I}} \coloneqq (G, I, O, \hat{B}, \omega, k, T)$ are equivalent in the sense that any solution set for one instance can be transformed to a solution set to the other instance with at least the same weights. This justifies solving $\hat{\mathcal{I}}$ instead of $\mathcal{I}$. Consider the boundary complementation $\mathcal{I}' \coloneqq (G', I', O', B', \omega', k')$ and $\hat{\mathcal{I}}' \coloneqq (G', I', O', \hat{B}', \omega', k')$ that are caused by the same (X, Y, Z, R). To show the claim, we consider the two directions. Any solution for $\hat{\mathcal{I}}'$ is immediately a solution for the same boundary complementation for $\mathcal{I}'$ since the only difference is that we limit the possible neighborhood to a subset of B.

For the other direction, consider a solution S to $\mathcal{I}'$, and we want to show that there is a solution $\hat{S}$ to $\hat{\mathcal{I}}'$ using only vertices of $\hat{B}'$ in the neighborhood with $\omega(\hat{S}) \geq \omega(S)$. If $S \cap W = \emptyset$, then S is also immediately a solution for $\hat{\mathcal{I}}'$. Therefore, assume $S \cap W \neq \emptyset$. Define $\tilde{X} \coloneqq \tilde{T} \cap S$, $\tilde{Y} \coloneqq \tilde{T} \cap \mathrm{N}_{G-(W \setminus U)}(S)$, and $\tilde{Z} \coloneqq \tilde{T} \setminus (\tilde{X} \cup \tilde{Y})$. Let $\tilde{R}$ be the set of $(a, b) \in \tilde{X} \times \tilde{X}$ such that there is an a-b-path in $G[S \setminus W \setminus U]$. Finally, let $\tilde{k} \coloneqq |\mathrm{N}(S) \cap W|$. Thus, we can construct a boundary complementation instance $(\tilde{G}', \tilde{I}', \tilde{O}', \tilde{B}', \tilde{w}', \tilde{k})$ from $(\tilde{X}, \tilde{Y}, \tilde{Z}, \tilde{R})$. One can easily verify that $(S \cap \mathrm{V}(\tilde{G}')) \cup \left\{ u_r \;\middle|\; r \in \tilde{R} \right\}$ is a feasible solution to this instance. Furthermore, the maximum solution $\tilde{S} \in \mathcal{R}$ to this instance gives a new set $\hat{S} \coloneqq (\tilde{S} \cap W) \cup (S \cap U) \subseteq \mathrm{V}(G')$ that has at least the same weight as S. We also know that $\mathrm{N}(\hat{S}) \subseteq \hat{B}'$ and $\left|\mathrm{N}(\hat{S})\right| \leq k$. See Figure 2 for a visualization of the construction corresponding to a solution S.

All that remains is to verify that $\hat{S}$ is strongly connected. For $v_1, v_2 \in \tilde{S} \cap W$, we can simply use the v_1-v_2-path in $\tilde{S}$, replacing subpaths via u_r for $r \in \tilde{R}$ with the actual paths in $S \cap U$. If $v_1 \in \tilde{S} \cap W$ and $v_2 \in S \cap U$, we can walk to any $x \in \tilde{X}$ which must have a path to v_2 in S, in which may need to replace subpaths via W. This can be done, since $\tilde{S} \cap W$ connects all pairs of $x_1, x_2 \in X$ that are not connected via $S \cap U$. The two remaining cases follow by symmetry, justifying the replacement of B with $\hat{B}$.

Finally, we have to show that the recursion terminates for the right choice of c; that is, both recursively solved instances have strictly smaller sizes than the original graph. For the first recursive call, note that the boundary complementation adds at most $k^2 \leq q$ vertices to $\tilde{G}$. Since $|U \setminus W| > q$, we have $\left|\mathrm{V}(\tilde{G})\right| < |\mathrm{V}(G)|$.

For the second recursive call, since $\left|\tilde{T}\right| \leq 2k$ and by Lemma 9, we have $|\mathcal{N}| \leq 2k + (k+1)k2^{c_1 k^2} \leq 2^{c_2 k^2}$ for constants c_1 and c_2. After applying all our reduction rules, by Lemma 25 for a suitable choice of c_3 and c we get

$$|W^*| \leq |\mathcal{N}| + 2^{2^{|\mathcal{N}|+1}+|\mathcal{N}|^2}\left(2^{|\mathcal{N}|+1} + |\mathcal{N}|^2\right) \leq 2^{c_3 2^{|\mathcal{N}|}} \leq 2^{c_3 2^{2^{c_2 k^2}}} \leq 2^{2^{2^{ck^2}}} =: q.$$

Since before the reductions, we had $|W \setminus U| > q$, the reduced graph G^* also has fewer vertices than G. Therefore, this recursive call also makes progress and the recursion terminates.

Runtime. We follow the analysis of [3]. With Lemma 26, we can test if the undirected version of G is unbreakable in time $2^{k \log q + k} n^{\mathcal{O}(1)} \leq 2^{2^{2^{\mathcal{O}(k^2)}}} n^{\mathcal{O}(1)}$. If the graph turns out to be unbreakable, the algorithm of Theorem 12 solves it in time $2^{\mathcal{O}(k^2 \log q)} \leq 2^{2^{2^{\mathcal{O}(k^2)}}} n^{\mathcal{O}(1)}$. Executing the reduction rules takes polynomial time in n by Lemma 25.

All that is left to do is analyze the recursion. Let $n' \coloneqq |W|$. By the separation property, we know that $q < n' < n - q$. From the correctness section, we get $|\mathrm{V}(G^*)| \leq |\mathrm{V}(G)| - n' + q$. Note that the recursion stops when the original graph is (q,k)-unbreakable, which must be the case if $|\mathrm{V}(G)| \leq 2q$. We arrive at the recurrence

$$T(n) = \begin{cases} 2^{2^{2^{\mathcal{O}(k^2)}}}, & \text{for } n \leq 2q; \\ \left(\max_{q<n'<n-q} T(n'+k^2) + T(n-n'+q)\right) + 2^{2^{2^{\mathcal{O}(k^2)}}} n^{\mathcal{O}(1)}, & \text{otherwise.} \end{cases}$$

Notice that $2^{2^{2^{\mathcal{O}(k^2)}}}$ appears in every summand of the expanded recurrence and can be ignored here and multiplied later. Furthermore, $n^{\mathcal{O}(1)}$ is bounded from above by a convex polynomial. Therefore, it is enough to consider the extremes of the maximum expression. For $n' = q+1$, the first recursive call evaluates to $T(q+1+k^2) \leq T(2q) \leq 2^{2^{2^{\mathcal{O}(k^2)}}}$. The second call only introduces an additional factor of n. For $n' = n-q-1$, the second expression evaluates to $T(2q+1)$ which is clearly also bounded by $2^{2^{2^{\mathcal{O}(k^2)}}}$. Thus, we arrive at the final runtime of $2^{2^{2^{\mathcal{O}(k^2)}}} n^{\mathcal{O}(1)}$. ◀

Finally, we can use Boundaried Max TSSCS to solve TSSCS. Since in our original problem every vertex could be part of the solution or its neighborhood, we initially set $I \coloneqq O \coloneqq \emptyset$ and $B \coloneqq \mathrm{V}(G)$. Furthermore, we only want to consider the boundary complementation that changes nothing, which we achieve by setting $T \coloneqq \emptyset$.

▶ **Theorem 1.** *Total-Secluded Strongly Connected Subgraph is solvable in time* $2^{2^{2^{\mathcal{O}(k^2)}}} n^{\mathcal{O}(1)}$.

3 Secluded $\mathcal{F}$-Free subgraph and Secluded DAG

In this section, we show that Out-Secluded $\mathcal{F}$-Free Subgraph is W[1]-hard for almost all choices of $\mathcal{F}$, even if we enforce weakly connected solutions. Except for a few missing cases, we establish a complete dichotomy when Out-Secluded $\mathcal{F}$-Free Weakly Connected Subgraph (Out-Secluded $\mathcal{F}$-Free WCS) is hard and when it is FPT. This is a surprising

result compared to undirected graphs and shows that out-neighborhood behaves completely different. The same problem was studied before in undirected graphs, where it admits FPT-algorithms using recursive understanding [11] and branching on important separators [15]. For total neighborhood, both algorithms still work efficiently since the seclusion condition acts exactly the same as in undirected graphs. However, the result changes when we look at out-neighborhood, where we show hardness in Theorem 2 for almost all choices of $\mathcal{F}$. To formalize for which kind of $\mathcal{F}$ the problem becomes W[1]-hard, we need one more definition.

▶ **Definition 28** (Inward Star). *We say that a directed graph G is an* inward star*, if there is one vertex $v \in \mathrm{V}(G)$ such that $\mathrm{E}(G) = \{(u, v) \mid u \in \mathrm{V}(G) \setminus \{v\}\}$, that is, the underlying undirected graph of G is a star and all edges are directed towards the center.*

Due to the nature of the construction, we show that if no inward star contains any $F \in \mathcal{F}$ as a subgraph, the problem becomes W[1]-hard. Besides this restriction, the statement holds for all non-empty $\mathcal{F}$, even families with only a single forbidden induced subgraph. Later, we explain how some other cases of $\mathcal{F}$ can be categorized as FPT or W[1]-hard.

▶ **Theorem 2.** *Let $\mathcal{F}$ be a non-empty set of directed graphs such that no $F \in \mathcal{F}$ is a subgraph of an inward star. Then, Out-Secluded $\mathcal{F}$-Free Weakly Connected Subgraph is W[1]-hard with respect to the parameter $k + w$ for unit weights.*

Proof Sketch. Let $F \in \mathcal{F}$. We give a reduction from Clique, inspired by [9]. Given an undirected graph G, k, and w, consider the incidence graph, which we modify in the following. We add $k + 2$ copies of F and a single vertex s. For every edge $e = \{u, v\} \in E$, we add the edges (e, u), (e, v), and (e, s). Furthermore, we add an edge from every vertex $v \in V_V$ to every vertex in every copy of F. For two different copies F_1 and F_2 of F, we add edges in both directions between every vertex $v_{f1} \in F_1$ and v_{f2} in F_2. Denote this newly constructed graph by G'. Finally, set $k' \coloneqq k$ and $w \coloneqq \binom{k}{2} + 1$.

The proof follows by showing that a clique of size k corresponds exactly to an out-secluded F-free weakly connected subgraph in G' with parameters k' and w. More details can be found in the full version [21]. ◀

Note that Theorem 2 is very general and excludes many properties Π immediately from admitting FPT-algorithms. Another interesting example is finding secluded DAGs, a very natural extension of secluded trees, studied in [11, 7]. There we choose $\mathcal{F}$ to be the set of all directed cycles. Although this set is not finite, hardness follows from Theorem 2.

▶ **Corollary 3.** *Out-Secluded Weakly Connected DAG is W[1]-hard with parameter $k + w$ for unit weights.*

Finally, we analyze some of the more remaining cases in the following paragraphs.

Trivial Cases. If $\mathcal{F}$ contains the graph with only a single vertex, the empty set is the only $\mathcal{F}$-free solution. If $\mathcal{F}$ contains two vertices connected by a single edge, weakly connected solutions can only consist of a single vertex or contain bidirectional edges (u, v) and (v, u) for $u, v \in \mathrm{V}(G)$. Both of these problems are clearly solvable in FPT-time, the second one via our algorithm for Secluded Clique. Additionally, if $\mathcal{F} = \emptyset$, we can clearly choose the maximum weight component of G as the solution.

Independent Sets. An independent set is a subgraph of an inward star. One kind of graph that cannot be included in $\mathcal{F}$ such that Theorem 2 shows hardness are independent sets. Surprisingly, the problem becomes FPT if $\mathcal{F}$ includes an independent set of any size. First, notice that independent sets that are part of $\mathcal{F}$ can be ignored if there is a smaller independent set in $\mathcal{F}$. Suppose the size of the smallest independent set is $\alpha + 1$. This means that any solution must be an α-bounded graph, i.e., a graph without an independent set of size α.

We have shown already how these can be enumerated efficiently with the algorithm in the proof of Theorem 4. For every enumerated subgraph, we can simply check if it is $\mathcal{F}$-free in time $|\mathcal{F}|\, n^{\|\mathcal{F}\|}$. Therefore, this case is also FPT with parameter k if $\mathcal{F}$ is finite.

▶ **Theorem 29.** *Let $\mathcal{F}$ be a finite family of directed graphs that contains an independent set. Then* Out-Secluded $\mathcal{F}$-Free WCS *is solvable in time $|\mathcal{F}|\,(2\alpha+2)^k n^{\|\mathcal{F}\|+\alpha+\mathcal{O}(1)}$.*

Inward Stars. Consider what happens when $\mathcal{F}$ contains an inward star F with two leaves. Then, the weakly connected F-free graphs are exactly the rooted trees where the root could be a cycle instead of a single vertex, with potentially added bidirectional edges. For $F \in \mathcal{F}$, we do not have a solution, but we conjecture that the FPT branching algorithm for Secluded Tree by [7] can be transferred to the directed setting.

If $\mathcal{F} = \{F\}$ for a star F with more than two leaves, the problem remains W[1]-hard. We can slightly modify the construction in the proof of Theorem 2 such that there is not one extra vertex s, but one extra vertex s_e for every $e \in \mathrm{E}(G)$. We connect all s_e internally in a directed path. A solution for Out-Secluded $\mathcal{F}$-Free WCS can then include the whole extra path and the edges encoding the clique. This avoids inward star structures with more than two leaves, showing hardness in this case. For only one single forbidden induced subgraph F, we therefore conjecture, that Out-Secluded $\mathcal{F}$-Free WCS is FPT with parameter k if and only if F has no edges or is an inward star with at most two leaves. For all other cases, we have proven W[1]-hardness with parameter $k + w$ for unit weights.

Remaining Cases. The previous paragraphs resolve almost all possible cases for $\mathcal{F}$, except for a few cases which are difficult to characterize. For example, we could have $\{F_s, F_p\} \subseteq \mathcal{F}$, where F_s is an inward star and F_p is a path, which makes a new connecting construction instead of s and the s_e necessary. The above characterization is still enough to give an understanding of which cases are hard and which are FPT for all natural choices of $\mathcal{F}$.

4 Secluded Subgraphs with Small Independence Number

In this section, we consider a generalization of the undirected Secluded Clique problem to directed graphs. We generalize this property in the following way.

▶ **Definition 30** (α-Bounded). *A directed graph G is called* α-bounded *if G includes no independent set of size $\alpha + 1$, that is, for all $S \subseteq \mathrm{V}(G)$ with $|S| = \alpha + 1$, the graph $G[S]$ includes at least one edge.*

Our problem of interest will be the Out-Secluded α-Bounded Subgraph problem (Out-Secluded α-BS). Note that α is part of the problem and therefore a constant. In the directed case, $\alpha = 1$ is analogous to the Out-Secluded Tournament problem, except that tournaments cannot include more than one edge between a pair of vertices.

In this section, we show how to solve Out-Secluded α-BS with a branching algorithm. The general nature of these branching rules allows us to transfer and optimize them, to significantly improve the best known runtime for the Secluded Clique problem to $1.6181^k n^{\mathcal{O}(1)}$ in the full version.

Our branching algorithm starts by selecting one part of the solution and then relies on the fact that the remaining solution has to be close around the selected part. More concretely, we start by picking an independent set that is part of the solution and build the remaining solution within its two-hop neighborhood. The following lemma justifies this strategy. The proof of this lemma was sketched on *Mathematics Stack Exchange* [14], we give an inspired proof again for completeness.

▶ **Lemma 31.** *In every directed graph G, there is an independent set $U \subseteq \mathrm{V}(G)$, such that every vertex in $\mathrm{V}(G) \setminus U$ is reachable from an $u \in U$ via a path of length at most 2.*

Proof. We prove the statement by induction on $|\mathrm{V}(G)|$. For $|\mathrm{V}(G)| = 1$, it clearly holds. Assume the statement holds for all graphs with fewer than $|\mathrm{V}(G)|$ vertices, we want to prove it for G. Let $v \in \mathrm{V}(G)$ be a vertex. If v reaches every other vertex via at most 2 edges, $\{v\}$ is our desired independent set. Otherwise, $T \coloneqq \mathrm{V}(G) \setminus \mathrm{N}^+[v]$ is non-empty. Since $|T| < |\mathrm{V}(G)|$, we can apply the induction hypothesis. So, let $T' \subseteq T$ be an independent set in $G[T]$ that reaches every vertex of T via at most two edges. We consider the edges between v and T'.

Since $T \cap \mathrm{N}^+(v) = \emptyset$, there cannot be an edge (v, t') for any $t' \in T'$. If there is an edge (t', v) for some $t' \in T'$, then T' is also a solution for G since t' can reach $\mathrm{N}^+[v]$ via at most 2 edges. Finally, if there is no edge between v and T', then $T' \cup \{v\}$ is an independent set. Also $T' \cup \{v\}$ reaches by definition all of $T \cup \mathrm{N}^+[v]$ via at most 2 edges. ◀

For α-bounded graphs, all independent set have size at most α, so trying every subset of $\mathrm{V}(G)$ of size at most α allows us to find a set that plays the role of U in Lemma 31 in our solution. Hence, the first step of our algorithm will be iterate over all such sets. From then on, we only consider solutions $S \subseteq \mathrm{N}^+[\mathrm{N}^+[U]]$.

Then, our algorithm uses two branching rules. Both rules have in common that in contrast to many well-known branching algorithms [5, Chapter 3], we never include vertices into a partial solution. Instead, due to our parameter, we only decide that vertices should be part of the final neighborhood. This means that we remove the vertex from the graph and decrease the parameter by 1. Our first branching rule branches on independent sets of size $\alpha + 1$ in $\mathrm{N}^+[\mathrm{N}^+[U]]$, to ensure that $\mathrm{N}^+[\mathrm{N}^+[U]]$ becomes α-bounded. The second rule is then used to decrease the size of the neighborhood of $\mathrm{N}^+[\mathrm{N}^+[U]]$, until it becomes secluded.

To formalize which vertices exactly to branch on, we use some new notation. For a vertex $v \in \mathrm{V}(G)$ and a vertex set $U \subseteq \mathrm{V}(G)$, pick an arbitrary shortest path from U to v if there exists one. Define $P_U(v)$ to be the vertices on that path including v but excluding the first vertex $u \in U$. Note that after removing vertices from the graph, $P_U(v)$ might change.

Now, we can give our algorithm for finding secluded α-bounded graphs.

▶ **Theorem 4.** *Out-Secluded α-Bounded Subgraph is solvable in time $(2\alpha+2)^k n^{\alpha+\mathcal{O}(1)}$.*

Proof. Let (G, ω, w, k) be an instance of Out-Secluded α-BS. We guess a vertex set $U \subseteq \mathrm{V}(G)$ that should be part of a desired solution S. Furthermore, we want to find a solution S to the instance such that $S \subseteq \mathrm{N}^+[\mathrm{N}^+[U]]$, that is, every $v \in S$ should be reachable from U via at most two edges. We give a recursive branching algorithm that finds the optimal solution under these additional constraints. The algorithm is also described in Algorithm 2.

Algorithm 2 The branching algorithm for Out-Secluded α-BS that returns a solution including the set $U \subseteq V(G)$.

```
def α-BS(G, ω, w, k, U):
    if k < 0 then
        abort
    else if N⁺[N⁺[U]] is a solution then
        return N⁺[N⁺[U]]
    else if there is an independent set I ⊆ N⁺[N⁺[U]] of size |I| = α + 1 then
        foreach v ∈ ⋃_{w∈I} P_U(w) do
            Call α-BS(G − v, ω, w, k − 1, U)
        end
    else if there is w ∈ N⁺(N⁺[N⁺[U]]) then
        foreach v ∈ P_U(w) do
            Call α-BS(G − v, ω, w, k − 1, U)
        end
    end
```

When deciding that a vertex should be part of the final neighborhood, we can simply delete it and decrease k by one. In case k decreases below 0, there is no solution. If $N^+[N^+[U]]$ is a solution to the instance, we return it. Otherwise we apply the following branching rules and repeat the algorithm for all non-empty independent sets $U \subseteq V(G)$ of size at most α.

Case 1. $N^+[N^+[U]]$ is not α-bounded. In this case, there must be an independent set $I \subseteq N^+[N^+[U]]$ of size $\alpha + 1$. Clearly, not all of I can be part of the solution, so there is a vertex $w \in I \setminus S$. This means that either w or a vertex on every path from U to w must be in $N^+(S)$. The set $P_U(w)$ is one such path of length at most 2. Thus, one of $\bigcup_{w \in I} P_U(w)$ must be part of the out-neighborhood of S and we branch on all of these vertices. For one vertex, delete it and decrease k by 1.

Case 2. $N^+[N^+[U]]$ is α-bounded, but has additional neighbors. Consider one of these neighbors $w \in N^+(N^+[N^+[U]])$. Since w is not reachable from U via at most two edges, we should not include it in the solution. Now, $P_U(w)$ is a path of length 3, and one of its vertices must be in $N^+(S)$. We again branch on all options, delete the corresponding vertex and decrease k by 1.

By Lemma 31, there is a suitable choice of U for every solution. Therefore, if we can find the maximum solution containing U if one exists in every iteration with our branching algorithm, the total algorithm is correct. Also notice that the branching rules are a complete case distinction; if none of the rules apply, the algorithm reaches a base case. The remaining proof of correctness follows from a simple induction.

We initially have to consider all subsets of $V(G)$ of size at most α while rejecting subsets that are not an independent set in time $n^{\alpha+1}$. To bound the runtime of the branching algorithm, notice that in each case we branch and make progress decreasing k by 1. In the first cases, the independent set I has size $\alpha + 1$ for every $w \in I$, the path $P_U(w)$ includes at most 2 vertices. Hence, there are at most $2(\alpha + 1)$ branches in this case. The second case gives exactly 3 branches and is thus dominated by the first rule. This gives the claimed runtime and concludes the proof. ◀

References

1 Shiri Chechik, Matthew P Johnson, Merav Parter, and David Peleg. Secluded connectivity problems. *Algorithmica*, 79:708–741, 2017. doi:10.1007/s00453-016-0222-z.

2 Jianer Chen, Yang Liu, Songjian Lu, Barry O'sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *Journal of the ACM*, 55:1–19, 2008. doi:10.1145/1411509.1411511.

3 Rajesh Chitnis, Marek Cygan, Mohammad Taghi Hajiaghayi, Marcin Pilipczuk, and Michał Pilipczuk. Designing fpt algorithms for cut problems using randomized contractions. *SIAM Journal on Computing*, 45:1171–1229, 2016. doi:10.1137/15M1032077.

4 Rajesh Chitnis, MohammadTaghi Hajiaghayi, and Dániel Marx. Fixed-parameter tractability of directed multiway cut parameterized by the size of the cutset. *SIAM Journal on Computing*, 42:1674–1696, 2013. doi:10.1137/12086217X.

5 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.

6 Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. Minimum bisection is fixed-parameter tractable. *SIAM Journal on Computing*, 48:417–450, 2019. doi:10.1137/140988553.

7 Huib Donkers, Bart M.P. Jansen, and Jari J.H. de Kroon. Finding k-secluded trees faster. *Journal of Computer and System Sciences*, 138:103461, 2023. doi:10.1016/j.jcss.2023.05.006.

8 Fedor V. Fomin, Petr A. Golovach, Nikolay Karpov, and Alexander S. Kulikov. Parameterized complexity of secluded connectivity problems. *Theory of Computing Systems*, 61:795–819, 2017. doi:10.1007/s00224-016-9717-x.

9 Fedor V Fomin, Petr A Golovach, and Janne H Korhonen. On the parameterized complexity of cutting a few vertices from a graph. In *Mathematical Foundations of Computer Science*, pages 421–432, 2013. doi:10.1007/978-3-642-40313-2_38.

10 Alexandra Fradkin and Paul Seymour. Edge-disjoint paths in digraphs with bounded independence number. *Journal of Combinatorial Theory, Series B*, 110:19–46, 2015. doi:10.1016/j.jctb.2014.07.002.

11 Petr A. Golovach, Pinar Heggernes, Paloma T. Lima, and Pedro Montealegre. Finding connected secluded subgraphs. *Journal of Computer and System Sciences*, 113:101–124, 2020. doi:10.1016/j.jcss.2020.05.006.

12 Martin Grohe, Ken-ichi Kawarabayashi, Dániel Marx, and Paul Wollan. Finding topological subgraphs is fixed-parameter tractable. In *Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing (STOC 2011)*, pages 479–488, 2011. doi:10.1145/1993636.1993700.

13 Meike Hatzel, Lars Jaffke, Paloma T Lima, Tomáš Masařík, Marcin Pilipczuk, Roohani Sharma, and Manuel Sorge. Fixed-parameter tractability of directed multicut with three terminal pairs parameterized by the size of the cutset: twin-width meets flow-augmentation. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3229–3244, 2023. doi:10.1137/1.9781611977554.ch123.

14 Sera Gunn (https://math.stackexchange.com/users/437127/sera gunn). Independent set of vertices in a directed graph. Mathematics Stack Exchange, 2017. URL: https://math.stackexchange.com/q/2292101.

15 Bart M. P. Jansen, Jari J. H. de Kroon, and Michał Włodarczyk. Single-exponential FPT algorithms for enumerating secluded f-free subgraphs and deleting to scattered graph classes. In *34th International Symposium on Algorithms and Computation (ISAAC 2023)*, pages 42:1–42:18, 2023. doi:10.4230/LIPIcs.ISAAC.2023.42.

16 Daniel Lokshtanov, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. Reducing CMSO Model Checking to Highly Connected Graphs. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, volume 107, pages 135:1–135:14, 2018. doi:10.4230/LIPIcs.ICALP.2018.135.

17 Max-Jonathan Luckow and Till Fluschnik. On the computational complexity of length-and neighborhood-constrained path problems. *Information Processing Letters*, 156:105913, 2020. doi:10.1016/j.ipl.2019.105913.

18 Pranabendu Misra, Saket Saurabh, Roohani Sharma, and Meirav Zehavi. Sub-exponential time parameterized algorithms for graph layout problems on digraphs with bounded independence number. *Algorithmica*, 85:2065–2086, 2023. doi:10.1007/s00453-022-01093-w.

19 Marcin Pilipczuk and Magnus Wahlström. Directed multicut is W[1]-hard, even for four terminal pairs. *ACM Transactions on Computation Theory*, 10:1–18, 2018. doi:10.1145/3201775.

20 Abhishek Sahu and Saket Saurabh. Kernelization of arc disjoint cycle packing in α-bounded digraphs. *Theory of Computing Systems*, 67:221–233, 2023. doi:10.1007/s00224-022-10114-8.

21 Jonas Schmidt, Shaily Verma, and Nadym Mallek. A parameterized study of secluded structures in directed graphs. *arXiv preprint arXiv:2502.06048*, 2025. doi:10.48550/arXiv.2502.06048.

22 René van Bevern, Till Fluschnik, George B. Mertzios, Hendrik Molter, Manuel Sorge, and Ondřej Suchý. The parameterized complexity of finding secluded solutions to some classical optimization problems on graphs. *Discrete Optimization*, 30:20–50, 2018. doi:10.1016/j.disopt.2018.05.002.

23 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.

Reforming an Unfair Allocation by Exchanging Goods

Sheung Man Yuen
National University of Singapore, Singapore

Ayumi Igarashi
University of Tokyo, Japan

Naoyuki Kamiyama
Kyushu University, Fukuoka, Japan

Warut Suksompong
National University of Singapore, Singapore

Abstract

Fairly allocating indivisible goods is a frequently occurring task in everyday life. Given an initial allocation of the goods, we consider the problem of reforming it via a sequence of exchanges to attain fairness in the form of envy-freeness up to one good (EF1). We present a vast array of results on the complexity of determining whether it is possible to reach an EF1 allocation from the initial allocation and, if so, the minimum number of exchanges required. In particular, we uncover several distinctions based on the number of agents involved and their utility functions. Furthermore, we derive essentially tight bounds on the worst-case number of exchanges needed to achieve EF1.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory; Theory of computation → Design and analysis of algorithms; Theory of computation → Problems, reductions and completeness

Keywords and phrases fair division, indivisible goods, envy-freeness, exchanges

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.54

Related Version *Full Version*: `https://arxiv.org/pdf/2412.19264` [34]

Funding This work was partially supported by the Singapore Ministry of Education under grant number MOE-T2EP20221-0001, by JST PRESTO under grant number JPMJPR20C1, by JSPS KAKENHI under grant number JP20H05795, by JST ERATO under grant number JPMJER2301, and by an NUS Start-up Grant.

Acknowledgements We would like to thank the anonymous reviewers for their valuable comments.

1 Introduction

Fair division is a research area that studies how to allocate scarce resources to interested agents in a fair manner. Over the past several decades, a large body of work has developed on concepts and algorithms for fairly allocating various types of resources [6, 27, 28, 30]. The developed theory has also been applied to many allocation scenarios in practice [9, 17, 19, 23].

A ubiquitous fair division problem is the allocation of *indivisible goods*, such as books, furniture, paintings, and human resources. While numerous fairness notions have been proposed for allocating indivisible goods, one of the most prominent notions is *envy-freeness up to one good (EF1)*. This notion requires that if an agent envies another agent, there must exist a good in the latter agent's bundle whose removal would make the envy disappear. Besides its simplicity, a salient feature of EF1 is that an allocation satisfying it always exists and can be found, e.g., by the *round-robin algorithm*, which lets the agents pick their favorite goods in a round-robin fashion. The allocation returned by this algorithm is also *balanced*, meaning that the numbers of goods that any pair of agents receive differ by at most one.

© Sheung Man Yuen, Ayumi Igarashi, Naoyuki Kamiyama, and Warut Suksompong;
licensed under Creative Commons License CC-BY 4.0

The fair division literature typically assumes that there is a set of unallocated goods and the objective is to allocate them fairly. In certain scenarios, however, an existing (possibly unfair) allocation of the goods is already in place, and the goal is to "reform" it in order to arrive at a fair allocation. This is the case, for instance, in the allocation of personnel among teams in an organization. As the personnel experience individual growth or decline, and as the needs of the teams evolve, these changes can necessitate a reevaluation and potential reformation of the current allocation by the organization leadership. Another example is the distribution of a museum's exhibits across its branches – the museum director may decide to adjust the distribution so as to ensure fairness based on the most recent valuations. Such scenarios fall under the umbrella of *control problems*, which assume a central authority with the power to control the outcome and have been studied extensively in fair division and computational social choice [11, 15].

In this work, we shall allow agents to *exchange* pairs of goods in the reformation process, and use EF1 as our fairness criterion. Exchanges constitute a fundamental type of operation and preserve the size of each agent's bundle, thereby ensuring that any cardinality constraints remain fulfilled.[1] Naturally, given an initial allocation, we wish to reach an EF1 allocation using a small number of exchanges. However, it is sometimes impossible to reach an EF1 allocation via *any* finite number of exchanges (e.g., if one agent receives many more goods than another agent in the initial allocation), so we start by exploring whether the corresponding decision problem can be answered efficiently. Since this problem is equivalent to determining whether an EF1 allocation with a certain size vector exists in a given instance, it is also meaningful independently of exchange considerations.[2] We also investigate other important questions in this setting. Namely, if it is possible to reach an EF1 allocation, can we efficiently determine the smallest number of exchanges required to achieve this goal? And how many exchanges might we need to make in the worst case in order to attain EF1?

1.1 Our Results

In our model, there is an initial allocation of a set of goods. As is often assumed in fair division, each agent has an additive utility function over the goods. At each step of the reformation process, two agents can exchange a pair of goods with each other to obtain another allocation, and the goal is to reach an EF1 allocation at the end of the process. More details on our model are provided in Section 2.

In Section 3, we consider the decision problem of determining whether a given initial allocation can be reformed into an EF1 allocation. As mentioned earlier, this problem is equivalent to determining whether an EF1 allocation with a given size vector exists, so we focus on the latter decision problem instead. We demonstrate interesting distinctions in the complexity based on the number of agents and their utility functions. Specifically, in the case of two agents, the problem can be solved in polynomial time if the agents have identical utilities, but becomes (weakly) NP-complete otherwise. For three or more agents, the problem is NP-complete even with identical utilities; however, it can be solved efficiently when the utilities are binary[3] provided that the number of agents is constant. Finally, for

[1] Capacity constraints are prevalent in fair division applications and have accordingly received interest in the literature [4, 21, 32, 33].

[2] A *size vector* specifies the number of goods that each agent receives in an allocation. When an EF1 allocation is not guaranteed to exist in some instances due to cardinality requirements, an approach taken by previous work is to relax the EF1 condition (e.g., [33]). However, this leads to unnecessarily weak guarantees in instances where EF1 can be attained.

[3] This means that each agent has utility 0 or 1 for each good.

Table 1 Computational complexity of REFORMABILITY, the problem of deciding whether an EF1 allocation with a given size vector exists in a given instance. The top row represents the class of utility functions considered. The leftmost column represents the number of agents. "sNP-c" and "wNP-c" stand for strongly NP-complete and weakly NP-complete respectively.

utilities	general	identical	binary	identical & binary
$n = 2$	wNP-c (Th. 6)	P (Th. 5)	P (Th. 10)	P (Th. 10)
constant $n \geq 3$	wNP-c (Th. 8)	wNP-c (Th. 8)	P (Th. 10)	P (Th. 10)
general n	sNP-c (Th. 11)	sNP-c (Th. 11)	NP-c (Th. 12)	P (Th. 13)

an arbitrary (non-constant) number of agents, the problem is strongly NP-hard even for identical utilities and NP-hard for binary utilities,[4] but can be solved in polynomial time if the utilities are identical *and* binary. The results of this section are summarized in Table 1.

Having determined the reformability of the initial allocation, we next explore in Section 4 the problem of deciding whether the optimal (i.e., minimum) number of exchanges required to reach an EF1 allocation is at most some given number k. For (a) two agents with identical utilities, (b) a constant number of agents with binary utilities, and (c) any number of agents with identical binary utilities, we show that this problem can be solved in polynomial time, just like the corresponding decision problem in Section 3. For the remaining scenarios, since deciding whether an allocation is reformable is already NP-complete (from Section 3), we instead focus on the special case where the allocation is balanced – an EF1 allocation is guaranteed to be reachable in this case (see Proposition 2). We show that the problem for this special case remains NP-complete.

Finally, in Section 5, instead of considering specific instances, we derive worst-case bounds on the number of exchanges necessary to reach an EF1 allocation. We assume that each of the n agents possesses s goods – this again ensures that an EF1 allocation is reachable. We show that roughly $s(n-1)/2$ exchanges always suffice for instances with general utilities or with binary utilities; moreover, our bound is essentially tight for any n and s, and exactly tight when $n = 2$ as well as when s is divisible by n. For instances with identical binary utilities, we show that an essentially tight bound for the number of exchanges is $sn/4$ for even n and $s(n-1)(n+1)/4n$ for odd n.

All omitted proofs can be found in the full version of our paper [34].

1.2 Additional Related Work

As mentioned earlier, the majority of work in fair division assumes that there is no initial allocation of the resources – we now discuss the key exceptions and their differences from our model. Boehmer et al. [5] studied the problem of *discarding* goods from an initial allocation in order to reach an envy-free or EF1 allocation. As it is possible to reallocate the goods in several practical situations, discarding them can be unnecessarily wasteful for the agents involved. In a similar vein, Dorn et al. [14] investigated deleting goods to attain another fairness notion called *proportionality*; they assumed that agents have ordinal preferences (rather than cardinal utilities) over the goods, and considered both the settings with and without an initial allocation.[5] Instead of deleting goods, Bredereck et al. [8] allowed

[4] For binary utilities, there is no distinction between weak and strong NP-hardness.

[5] When there is no initial allocation, Dorn et al. [14] considered deleting goods so that a proportional allocation of the remaining goods exists. Aziz et al. [2] examined discarding or adding goods to achieve envy-freeness, also in the absence of an initial allocation and under ordinal preferences.

agents to share goods in order to improve their allocations, while Bentert et al. [3] analyzed the resolution of envy by adding goods. Aziz et al. [1] focused on reallocating goods to make agents better off, but did not delve into the aspect of fairness. Igarashi et al. [22] aimed to transition from an initial allocation to a target allocation, both of which are EF1, while maintaining EF1 throughout the process. Chandramouleeswaran et al. [10] examined *transferring* goods starting from a "near-EF1" allocation with the goal of reaching an EF1 allocation. Segal-Halevi [31] considered the reallocation of a *divisible* good and explored the trade-off between guaranteeing a minimum utility for every agent and ensuring each agent a certain fraction of her original utility. Chevaleyre et al. [12] also strived to reach fair allocations but via exchanges with money.

Further afield, the idea of improving an initial allocation has also been examined when each agent receives only one good, a setting sometimes known as a *housing market*. Gourvès et al. [18] assumed an underlying social network and allowed beneficial exchanges between agents who are neighbors in the network – their work led to a series of follow-up papers on similar models [20, 25, 26, 29]. Damamme et al. [13] also considered exchanges but without an underlying graph structure, while Brandt and Wilczynski [7] used *Pareto-optimality* as their target notion. The papers mentioned so far in this paragraph did not have fairness as their objective. Ito et al. [24] incorporated fairness in the form of envy-freeness into this setting – starting with an envy-free allocation, they let each agent exchange her current good with a preferred unassigned good as long as the exchange keeps the allocation envy-free.

2 Preliminaries

Let $N = \{1, \dots, n\}$ be a set of $n \geq 2$ agents, and M be a set of $m \geq 1$ goods typically denoted by $g_1, \dots, g_m$. A *bundle* is a (possibly empty) subset of goods. An *allocation* $\mathcal{A} = (A_1, \dots, A_n)$ is an ordered partition of M into n bundles such that bundle A_i is allocated to agent $i \in N$. An *(allocation) size vector* $\vec{s} = (s_1, \dots, s_n)$ is a vector of non-negative integers such that $\sum_{i \in N} s_i = m$. We say that an allocation $\mathcal{A}$ has size vector $\vec{s}$ if $|A_i| = s_i$ for all $i \in N$. A size vector $\vec{s}$ is *balanced* if $|s_i - s_j| \leq 1$ for all $i, j \in N$, and an allocation is *balanced* if it has a balanced size vector.

Each agent $i \in N$ has an additive *utility function* $u_i : 2^M \to \mathbb{R}_{\geq 0}$ that maps bundles to non-negative real numbers; additivity means that $u_i(M') = \sum_{g \in M'} u_i(\{g\})$ for all $i \in N$ and $M' \subseteq M$. We write $u_i(g)$ instead of $u_i(\{g\})$ for a single good $g \in M$. The utility functions are *identical* if $u_i = u_j$ for all $i, j \in N$ – we shall use u to denote the common utility function in this case. The utility functions are *binary* if $u_i(g) \in \{0, 1\}$ for all $i \in N$ and $g \in M$. Agent i is *EF1 towards* agent j in an allocation $\mathcal{A} = (A_1, \dots, A_n)$ if either $A_j = \varnothing$ or there exists a good $g \in A_j$ such that $u_i(A_i) \geq u_i(A_j \setminus \{g\})$. An allocation $\mathcal{A}$ is *EF1* if every agent is EF1 towards every other agent in $\mathcal{A}$. A *(fair division) instance* $\mathcal{I}$ consists of a set of agents N, a set of goods M, and the agents' utility functions $(u_i)_{i \in N}$.

An *exchange* involves an agent i giving one good from her bundle to agent j and simultaneously receiving one good from agent j's bundle. We say that allocations $\mathcal{A} = (A_1, \dots, A_n)$ and $\mathcal{B} = (B_1, \dots, B_n)$ can be *reached via an exchange* if there exist distinct $i, j \in N$, $g \in A_i$, and $g' \in A_j$ such that $B_i = (A_i \cup \{g'\}) \setminus \{g\}$, $B_j = (A_j \cup \{g\}) \setminus \{g'\}$, and $B_k = A_k$ for all $k \in N \setminus \{i, j\}$. An allocation $\mathcal{B}$ can be *reached* from an allocation $\mathcal{A}$ if there exist a non-negative integer T and a sequence of allocations $(\mathcal{A}^0, \mathcal{A}^1, \dots, \mathcal{A}^T)$ such that $\mathcal{A}^0 = \mathcal{A}$, $\mathcal{A}^T = \mathcal{B}$, and for each $t \in \{0, \dots, T-1\}$, $\mathcal{A}^t$ and $\mathcal{A}^{t+1}$ can be reached via an exchange. The *optimal number of exchanges* required to reach $\mathcal{B}$ from $\mathcal{A}$ is the smallest T

across all such sequences of allocations – if no such T exists (i.e., $\mathcal{B}$ cannot be reached from $\mathcal{A}$), then the optimal number of exchanges is defined to be ∞.[6] Observe that two allocations can be reached from each other if and only if they share the same size vector.

▶ **Proposition 1.** *Given an instance, let $\mathcal{A}$ and $\mathcal{B}$ be allocations in the instance. Then, $\mathcal{B}$ can be reached from $\mathcal{A}$ if and only if $\mathcal{A}$ and $\mathcal{B}$ have the same size vector.*

Next, we state a simple proposition that characterizes the existence of EF1 allocations based on the size vector.

▶ **Proposition 2.** *Let $\vec{s} = (s_1, \ldots, s_n)$, and let $m = \sum_{i=1}^{n} s_i$.*

(a) *If $\vec{s}$ is balanced, then every instance with n agents and m goods admits an EF1 allocation with size vector $\vec{s}$.*

(b) *If $\vec{s}$ is not balanced, then there exists an instance with n agents and m goods that does not admit any EF1 allocation with size vector $\vec{s}$.*

Finally, we introduce an NP-hard decision problem called the Balanced Multi-Partition problem, which we will use later in the proofs of several results. In Balanced Multi-Partition, we are given positive integers p, q, K and a multiset of positive integers $X = \{x_1, \ldots, x_{pq}\}$ such that $K < x_j \leq 2K$ for all $j \in \{1, \ldots, pq\}$, and the sum of all the integers in X is $p(q+1)K$. The problem is to decide whether X can be partitioned into multisets $X_1, \ldots, X_p$ of equal cardinalities and sums, i.e., for each $i \in \{1, \ldots, p\}$, the cardinality of X_i is q and the sum of all the integers in X_i is $(q+1)K$. The NP-hardness of this problem is based on a reduction from the equal-cardinality version of the NP-hard problem Partition [16, p. 223].

▶ **Proposition 3.** *For any fixed $p \geq 2$, Balanced Multi-Partition is NP-hard.*

3 Reformability of Allocations

We start by investigating the decision problem of whether a given initial allocation can be reformed into an EF1 allocation. By Proposition 1, this reformation is possible if and only if there exists an EF1 allocation with the same size vector as the initial allocation. Therefore, in the rest of this section, we shall equivalently focus on the problem of deciding the existence of an EF1 allocation with a given size vector – this problem can be of interest independently of reformation considerations, e.g., when space constraints are present.

Now, Proposition 2 tells us that an EF1 allocation with a balanced size vector always exists. This means that the only time when we may have difficulties in ascertaining whether an EF1 allocation exists is when the given size vector is *not* balanced. In fact, as some of our proofs in this section show, the decision problem is NP-hard even when the sizes of the agents' bundles differ by *exactly two* (e.g., in Theorem 6).

We discuss the cases of two agents, a constant number of agents, and a general number of agents separately. For each case, we explore how the hardness of the decision problem varies across different classes of utility functions. Our results are summarized in Table 1.

[6] The optimal number of exchanges can be viewed as the distance between the two allocations in the implicit *exchange graph*, where the allocations are vertices and the edges connect allocations that can be reached via an exchange [22]. When there is no path connecting two vertices of a graph, it is common to define the distance between them as infinity.

For convenience, we refer to as REFORMABILITY the problem of deciding whether an EF1 allocation with a given size vector exists in a given instance. Note that REFORMABILITY is in NP regardless of the number of agents, as we can verify in polynomial time whether a given allocation satisfies the condition by simply checking its size vector and comparing the bundles of every pair of agents for EF1.

3.1 Two Agents

For two agents, interestingly, the computational complexity of the problem turns out to be different depending on whether the agents have identical utilities or not. We begin our discussion with the case of identical utilities.

For two agents with identical utilities, we first provide a characterization for the existence of a desired EF1 allocation based on the size vector and the utilities of the goods. We show that an EF1 allocation with a given size vector exists if and only if the agent with fewer goods (say, agent 1) is EF1 towards the other agent (say, agent 2) in the allocation where agent 1 receives the most valuable goods. Note that the condition in the lemma only requires checking that agent 1 is EF1 towards agent 2; in particular, it does not require checking that agent 2 is EF1 towards agent 1.

▶ **Lemma 4.** *Given an instance with two agents with identical utilities, let $\vec{s} = (s_1, s_2)$ be a size vector with $s_1 \leq s_2$. Assume that the goods $g_1, \ldots, g_m$ are arranged in non-increasing order of utility, and let $M_0 = \{g_1, \ldots, g_{s_1}\}$. Then, there exists an EF1 allocation with size vector $\vec{s}$ if and only if agent 1 is EF1 towards agent 2 in the allocation $(M_0, M \setminus M_0)$.*

Proof. We say in this proof that for any nonempty set $M' \subseteq M$, the good $g_i \in M'$ is the *most valuable* good in M' if g_i is the good with the smallest index in M'; likewise, g_i is the *least valuable* good in M' if g_i is the good with the largest index in M'. Note that the most (resp. least) valuable good in M' is the one with the highest (resp. lowest) utility among all the goods in M', with ties broken by index.

($\Rightarrow$) Let (A_1, A_2) be an EF1 allocation with size vector $\vec{s}$. Let g and g' be the most valuable good in A_2 and $M \setminus M_0$ respectively. Since M_0 is the set containing the s_1 most valuable goods, we have $u(M_0) \geq u(A_1)$. Since (A_1, A_2) is an EF1 allocation, we have $u(A_1) \geq u(A_2 \setminus \{g\})$. Moreover, since $M \setminus M_0$ is the set containing the s_2 least valuable goods, we have $u(A_2 \setminus \{g\}) \geq u((M \setminus M_0) \setminus \{g'\})$. Combining the three inequalities, we get $u(M_0) \geq u((M \setminus M_0) \setminus \{g'\})$. It follows that agent 1 is EF1 towards agent 2 in the allocation $(M_0, M \setminus M_0)$.

($\Leftarrow$) Suppose that agent 1 is EF1 towards agent 2 in the allocation $(M_0, M \setminus M_0)$. If agent 2 is also EF1 towards agent 1 in $(M_0, M \setminus M_0)$, then we are done; therefore, assume that agent 2 envies agent 1 by more than one good. For notational simplicity, let $h_j = g_{s_1+j}$ for $j \in \{1, \ldots, s_1\}$, so that the goods arranged in non-increasing order of utility are $g_1, g_2, \ldots, g_{s_1}, h_1, h_2, \ldots, h_{s_1}, g_{2s_1+1}, \ldots, g_m$. Let $A_1^1 = M_0 = \{g_1, \ldots, g_{s_1}\}$ and $A_2^1 = M \setminus M_0 = \{h_1, \ldots, h_{s_1}\} \cup \{g_{2s_1+1}, \ldots, g_m\}$.

Let $t = 1$. In the allocation (A_1^t, A_2^t), agent 1 is EF1 towards agent 2, but agent 2 envies agent 1 by more than one good. Since g_t is the most valuable good in A_1^t, we have $u(A_2^t) < u(A_1^t \setminus \{g_t\})$. Let $A_1^{t+1} = (A_1^t \cup \{h_t\}) \setminus \{g_t\}$ and $A_2^{t+1} = (A_2^t \cup \{g_t\}) \setminus \{h_t\}$ be the bundles after exchanging g_t and h_t. Then, we have

$$\begin{aligned} u(A_1^{t+1}) &= u((A_1^t \cup \{h_t\}) \setminus \{g_t\}) \\ &\geq u(A_1^t \setminus \{g_t\}) > u(A_2^t) = u((A_2^{t+1} \cup \{h_t\}) \setminus \{g_t\}) \geq u(A_2^{t+1} \setminus \{g_t\}), \end{aligned}$$

so agent 1 is EF1 towards agent 2 in (A_1^{t+1}, A_2^{t+1}). If agent 2 is also EF1 towards agent 1, then (A_1^{t+1}, A_2^{t+1}) is an EF1 allocation and we are done. Otherwise, agent 2 envies agent 1 by more than one good, and we increment t by 1 and repeat the discussion in this paragraph.

If we still have not found an EF1 allocation after $t = s_1$, then agent 1 is EF1 towards agent 2 in $(A_1^{s_1+1}, A_2^{s_1+1})$, where $A_1^{s_1+1} = \{h_1, \ldots, h_{s_1}\} \subseteq A_2^1$ and $A_2^{s_1+1} = \{g_1, \ldots, g_{s_1}\} \cup \{g_{2s_1+1}, \ldots, g_m\} \supseteq A_1^1$, and g_1 is the most valuable good in $A_2^{s_1+1}$. This implies that

$$u(A_1^{s_1+1}) \leq u(A_2^1) < u(A_1^1 \setminus \{g_1\}) \leq u(A_2^{s_1+1} \setminus \{g_1\}),$$

which means that agent 1 is *not* EF1 towards agent 2 in $(A_1^{s_1+1}, A_2^{s_1+1})$. This is a contradiction; therefore, (A_1^t, A_2^t) must be EF1 for some $t \in \{1, \ldots, s_1\}$. ◀

Since the condition in Lemma 4 can be checked in polynomial time, we can derive the following result.

▶ **Theorem 5.** *Reformability is in P for two agents with identical utilities.*

While deciding whether an EF1 allocation with a given size vector exists can be done efficiently for two agents with identical utilities, we remark here that deciding whether an *envy-free* allocation exists is NP-hard for two agents with identical utilities even if we allow any size vector – this follows directly from a reduction from Partition.[7]

We now proceed to general utilities. Lemma 4 assumes identical utilities, and there is no obvious way to generalize it to non-identical utilities. In fact, perhaps surprisingly, we show that the decision problem becomes NP-hard when we drop the assumption of identical utilities. The proof follows from a reduction from Balanced Multi-Partition with $p = 2$, an NP-hard problem by Proposition 3.

▶ **Theorem 6.** *Reformability is weakly NP-complete for two agents.*

Proof. Clearly, this problem is in NP. The "weak" aspect is demonstrated later in Lemma 7, which says that there exists a pseudopolynomial-time algorithm that solves this problem for any constant number of agents. Therefore, it suffices to show that this problem is NP-hard.

To demonstrate NP-hardness, we shall reduce from the NP-hard problem Balanced Multi-Partition with $p = 2$ (see Proposition 3). Let a Balanced Multi-Partition instance be given with $p = 2$. Without loss of generality, assume that $q \geq 2$. Let $Y = \{y_1, \ldots, y_{2q+2}\}$ be a multiset such that $y_j = x_j$ for $j \in \{1, \ldots, 2q\}$, $y_{2q+1} = 2K$, and $y_{2q+2} = 0$. We claim that Y can be partitioned into two multisets Y_1 and Y_2 of equal cardinalities (i.e., of size $q+1$ each) with sums $(q+3)K$ and $(q+1)K$ respectively if and only if X can be partitioned into two multisets X_1 and X_2 of equal cardinalities and sums. If the latter condition holds, then let Y_1 (resp. Y_2) contain the corresponding elements in X_1 (resp. X_2), and let $y_{2q+1} \in Y_1$ and $y_{2q+2} \in Y_2$ – this gives an appropriate partition of Y. Conversely, if the former condition holds, then we show that X can be partitioned appropriately. Note that if $y_{2q+1} \in Y_2$, then there are at least $q - 1 > 0$ integers in $\{y_1, \ldots, y_{2q}\}$ that are also in Y_2. Since every integer in $\{y_1, \ldots, y_{2q}\}$ is more than K, the sum of Y_2 will be more than $(q-1)K + 2K = (q+1)K$, which is a contradiction. This means that $y_{2q+1} \in Y_1$. Similarly, if $y_{2q+2} \in Y_1$, then there are exactly $q+1$ integers in $\{y_1, \ldots, y_{2q}\}$ that are in Y_2. The sum of Y_2 will be more than $(q+1)K$, which is a contradiction. Hence, $y_{2q+2} \in Y_2$. Now, this means that $\{y_1, \ldots, y_{2q}\}$ must be partitioned into two multisets of equal cardinalities with sum $(q+1)K$ each. This induces an appropriate partition of X.

[7] If we require both agents to receive the same number of goods, the problem for envy-freeness remains NP-hard by a reduction from the equal-cardinality version of Partition.

Next, define a fair division instance as follows. There are $n = 2$ agents and a set of goods $M = \{g_1, \ldots, g_{2q+6}\}$. Agent 2's utility is such that $u_2(g_j) = y_j$ for $j \in \{1, \ldots, 2q+2\}$, $u_2(g_{2q+3}) = u_2(g_{2q+4}) = 0$, and $u_2(g_{2q+5}) = u_2(g_{2q+6}) = 2K$. Agent 1's utility is such that $u_1(g) = u_2(g) + 4K$ for all $g \in M$. The size vector $\vec{s}$ is $(q+2, q+4)$. This reduction can be done in polynomial time. We claim that there exists an EF1 allocation with size vector $\vec{s}$ in this instance if and only if Y can be partitioned into two multisets Y_1 and Y_2 of equal cardinalities (i.e., of size $q+1$ each) with sums $(q+3)K$ and $(q+1)K$ respectively.

($\Leftarrow$) Let J_1' and J_2' be a partition of $\{1, \ldots, 2q+2\}$ of equal cardinalities such that $\sum_{j \in J_1'} y_j = (q+3)K$ and $\sum_{j \in J_2'} y_j = (q+1)K$. Let $A_1 = \{g_j \mid j \in J_1'\} \cup \{g_{2q+5}\}$ and $A_2 = M \setminus A_1$ be the two agents' bundles respectively. From agent 1's perspective, agent 1's bundle has utility $((q+3)K + 2K) + (q+2)(4K) = (5q+13)K$, agent 2's bundle has utility $((q+1)K + 2K) + (q+4)(4K) = (5q+19)K$, and a most valuable good in agent 2's bundle (e.g., g_{2q+6}) has utility $6K$, so agent 1 is EF1 towards agent 2. From agent 2's perspective, agent 2's bundle has utility $(q+1)K + 2K = (q+3)K$, agent 1's bundle has utility $(q+3)K + 2K = (q+5)K$, and a most valuable good in agent 1's bundle (e.g., g_{2q+5}) has utility $2K$, so agent 2 is EF1 towards agent 1. Accordingly, (A_1, A_2) is an EF1 allocation with size vector $(q+2, q+4)$.

($\Rightarrow$) Let (A_1, A_2) be an EF1 allocation with size vector $\vec{s}$. From agent 1's perspective, $u_1(M) = (10q+32)K$ and a most valuable good (e.g., g_{2q+5}) has utility $6K$. For agent 1 to be EF1 towards agent 2, we must have $u_1(A_1) \geq ((10q+32)K - 6K)/2 = (5q+13)K$ and $u_2(A_1) = u_1(A_1) - (q+2)(4K) \geq (q+5)K$. This means that $u_1(A_2) = u_1(M) - u_1(A_1) \leq (5q+19)K$ and $u_2(A_2) = u_1(A_2) - (q+4)(4K) \leq (q+3)K$. On the other hand, from agent 2's perspective, $u_2(M) = (2q+8)K$ and a most valuable good has utility $2K$. For agent 2 to be EF1 towards agent 1, we must have $u_2(A_2) \geq ((2q+8)K - 2K)/2 = (q+3)K$ and $u_1(A_2) = u_2(A_2) + (q+4)(4K) \geq (5q+19)K$. This means that $u_2(A_1) = u_2(M) - u_2(A_2) \leq (q+5)K$ and $u_1(A_1) = u_2(A_1) + (q+2)(4K) \leq (5q+13)K$. By combining these inequalities, we conclude that these inequalities are tight, i.e., agent 1's utilities for both agents' bundles are *exactly* $(5q+13)K$ and $(5q+19)K$ respectively so that the sum is $(10q+32)K$, and agent 2's utilities for both agents' bundles are *exactly* $(q+5)K$ and $(q+3)K$ respectively so that the sum is $(2q+8)K$. Additionally, both agents must each have a most valuable good worth $6K$ and $2K$ to them respectively. Without loss of generality, we may assume that $g_{2q+5} \in A_1$ and $g_{2q+6} \in A_2$ (note that g_{2q+1} is also a most valuable good, but we use g_{2q+5} and g_{2q+6} for simplicity).

Since $g_{2q+6} \in A_2$, there are $q+3$ goods in $A_2 \setminus \{g_{2q+6}\}$ and $u_2(A_2 \setminus \{g_{2q+6}\}) = (q+3)K - 2K = (q+1)K$. These goods are chosen from $M_1 = \{g_1, \ldots, g_{2q+1}\}$ and $M_0 = \{g_{2q+2}, g_{2q+3}, g_{2q+4}\}$. Recall from the construction that $u_2(g) > K$ for all $g \in M_1$, and $u_2(g) = 0$ for all $g \in M_0$. If $A_2 \setminus \{g_{2q+6}\}$ contains at least $q+1$ goods from M_1, then $u_2(A_2 \setminus \{g_{2q+6}\}) > (q+1)K$, a contradiction. Therefore, $A_2 \setminus \{g_{2q+6}\}$ contains at most q goods from M_1, and at least 3 goods from M_0. Since $|M_0| = 3$, we must have $M_0 \subseteq A_2$. Note that $u_2(M_0) = 0$, so $u_2((A_2 \setminus \{g_{2q+6}\}) \setminus M_0) = (q+1)K$. Therefore, the q goods from M_1 in agent 2's bundle have a total utility of $(q+1)K$. These goods, together with g_{2q+2}, induce the set Y_2 with cardinality $q+1$ and sum $(q+1)K$. Then, $Y_1 = Y \setminus Y_2$ and Y_2 give a required partition of Y. ◀

The proof of Theorem 6 suggests that the problem is NP-hard even when the sizes of the two agents' bundles differ by *exactly two*. Note that this problem is in P when the sizes of the agents' bundles differ by *at most one* (in fact, every such instance is a Yes-instance by Proposition 2). For two agents with binary utilities, we shall show later that the decision problem is in P (see Theorem 10).

3.2 Constant Number of Agents

Next, we discuss the complexity of the decision problem for a *constant number of agents*. In this case, we devise a pseudopolynomial-time algorithm for deciding the existence of an EF1 allocation with a given size vector. This algorithm uses dynamic programming to check for such an allocation.

▶ **Lemma 7.** *Let an instance with n agents and a size vector be given, where n is a constant. Suppose that the utility of each good is an integer, and let $R = \max_{i \in N} u_i(M)$. Then, there exists an algorithm running in time polynomial in m and R that decides whether the instance admits an EF1 allocation with the size vector.*

Proof. The algorithm uses dynamic programming. We construct a table with m columns and L rows, where L will be specified later. The index of each row is represented by a tuple containing $a_{i,j}$, $b_{i,j}$, and c_i for each $i, j \in N$, i.e., $(a_{1,1}, a_{1,2}, \ldots, a_{n,n}, b_{1,1}, b_{1,2}, \ldots, b_{n,n}, c_1, \ldots, c_n)$. The value of $a_{i,j}$ is the utility of agent j's bundle from agent i's perspective, i.e., $a_{i,j} = u_i(A_j)$; the value of $b_{i,j}$ is the utility of a most valuable good in agent j's bundle from agent i's perspective, i.e., $b_{i,j} = \max_{g \in A_j} u_i(g)$ (note that this value is zero if $A_j = \varnothing$); and the value of c_i is the number of goods in agent i's bundle. Note that $a_{i,j}, b_{i,j} \in \{0, \ldots, R\}$ and $c_i \in \{0, \ldots, m\}$, so there are $L = (R+1)^{2n^2}(m+1)^n$ rows, which is polynomial in m and R. An entry in column q represents whether an allocation involving $\{g_1, \ldots, g_q\}$ is possible for the tuple representing the row, and is either positive or negative.

Initialize every entry to negative. Consider the n possibilities of adding g_1 to each of the agents' bundles respectively, and set the corresponding entries in the first column of the table to positive. In particular, for each $j \in N$, the row represented by the tuple such that $a_{i,j} = b_{i,j} = u_i(g_1)$ and $c_j = 1$ for all $i \in N$, and zero for all other values in the tuple, has the entry (in the first column) set to positive.

Now, for each $q \in \{2, \ldots, m\}$ in ascending order, for each positive entry in column $q-1$, consider the n possibilities of adding g_q into each of the n agents' bundles respectively, and set the corresponding entry for each of these possibilities in column q to positive. Once this procedure is done, consider all positive entries in column m. If some positive entry corresponds to an EF1 allocation with the required size vector, then the instance admits such an EF1 allocation; otherwise, no such allocation exists.

Since n is a constant, the number of entries in the table is polynomial in m and R. At each column, there is a polynomial number of rows with positive entries, and hence the update is polynomial. Finally, checking for a feasible EF1 allocation at the last column can also be done in polynomial time. ◀

We now move to *polynomial-time* algorithms that determine the existence of an EF1 allocation with a given size vector. Recall that such an algorithm exists for two agents with identical utilities (Theorem 5). However, it turns out that such an algorithm does not exist for three or more agents with identical utilities, unless P = NP. In particular, we establish the NP-hardness of the decision problem via a reduction from Balanced Multi-Partition with $p = 2$, an NP-hard problem by Proposition 3.

▶ **Theorem 8.** *Reformability is weakly NP-complete for $n \geq 3$ agents with identical utilities, where n is a constant.*

Proof. Clearly, this problem is in NP. The "weak" aspect is demonstrated in Lemma 7. Therefore, it suffices to show that this problem is NP-hard.

To show NP-hardness, we shall reduce from the NP-hard problem Balanced Multi-Partition with $p = 2$ (see Proposition 3). Let a Balanced Multi-Partition instance with $p = 2$ be given. Define a fair division instance as follows. There are $n \geq 3$ agents with

identical utilities, and a set of goods $M = \{g_1, \ldots, g_{2q}, h_1, \ldots, h_n\}$ such that $u(g_j) = x_j$ for $j \in \{1, \ldots, 2q\}$ and $u(h_k) = (q+1)K$ for $k \in \{1, \ldots, n\}$. The size vector $\vec{s}$ is such that $s_1 = s_2 = q+1$ and $s_k = 1$ for all $k \in \{3, \ldots, n\}$. This reduction can be done in polynomial time. We claim that there exists an EF1 allocation with size vector $\vec{s}$ in this instance if and only if X can be partitioned into multisets X_1, X_2 of equal cardinalities and sums.

($\Leftarrow$) Let (X_1, X_2) be such a partition. Define an allocation such that agent k receives h_k for $k \in N$, agent 1 additionally receives the q goods corresponding to the integers in X_1, and agent 2 additionally receives the q goods corresponding to the integers in X_2. We show that this allocation is EF1. The utilities of agent 1's and agent 2's bundles are $2(q+1)K$ each, and the utilities of the other agents' bundles are $(q+1)K$ each, so agents 1 and 2 do not envy anyone else. Therefore, it remains to check that agent k is EF1 towards agents 1 and 2 for $k \in \{3, \ldots, n\}$. Upon the removal of the single good h_1 (resp. h_2) from agent 1's (resp. agent 2's) bundle, the remaining bundle has utility $(q+1)K$, so agent k is EF1 towards agent 1 (resp. agent 2). Therefore, the allocation is EF1, as desired.

($\Rightarrow$) Let $(A_1, \ldots, A_n)$ be an EF1 allocation with size vector $(q+1, q+1, 1, \ldots, 1)$. If agent 1's bundle has at least two goods from $\{h_1, \ldots, h_n\}$, then her bundle without the most valuable good has utility more than $(q+1)K$ since her bundle also contains other goods with positive utility. Agent 3, having a bundle of utility at most $(q+1)K$, will not be EF1 towards agent 1, contradicting the assumption that the allocation is EF1. Therefore, agent 1's bundle has at most one good from $\{h_1, \ldots, h_n\}$; likewise for agent 2's bundle. This means that every agent receives *exactly* one good from $\{h_1, \ldots, h_n\}$. Having established this, agent 3's bundle has a utility of $(q+1)K$, and agent 3 is EF1 towards agent 1. This means that agent 1's bundle without a most valuable good (say, some h_k) must have utility at most $(q+1)K$. The same argument can be used to show the same statement for agent 2's bundle. This means that the goods $\{g_1, \ldots, g_{2q}\}$ must be divided between agents 1 and 2 with each agent receiving a utility of *exactly* $(q+1)K$. Such a division of $\{g_1, \ldots, g_{2q}\}$ induces a partition of X into two multisets of equal cardinalities and sums, as desired. ◀

Since the decision problem is NP-hard even for identical utilities, it must also be NP-hard for general utilities. We now consider another class of utilities: binary utilities. When there are n agents, every good g belongs to one of 2^n types of goods represented by the vector $(u_1(g), \ldots, u_n(g))$. For the purpose of determining whether an EF1 allocation exists, it suffices to consider different goods of the same type as *indistinguishable*. We say that two allocations are in the same equivalence class if the number of goods of each type that each agent has is the same in both allocations. If $\mathcal{A}$ is an EF1 allocation, then all allocations in the same equivalence class as $\mathcal{A}$ are also EF1 and are reachable from $\mathcal{A}$. We shall proceed with a result which enumerates all (essentially equivalent) EF1 allocations in time polynomial in the number of goods, provided that the number of agents is a constant.

▶ **Lemma 9.** *Let an instance with n agents with binary utilities and a size vector be given, where n is a constant. Then, there exists an algorithm running in time polynomial in m that outputs all equivalence classes of EF1 allocations with the size vector.*

Proof. An agent's bundle can be represented by a 2^n-vector where each component of the vector is the number of goods of that type in her bundle. Since the number of goods of each type is an integer between 0 and m, there are $m+1$ possible values for each component, and hence at most $(m+1)^{2^n}$ possible vectors to represent each agent's bundle. Allocations in an equivalence class can be represented by an ordered collection of n such vectors – one for each agent – and there are at most $((m+1)^{2^n})^n$ such collections. Since $((m+1)^{2^n})^n$ is polynomial in m whenever n is a constant, there is at most a polynomial number of possible

equivalence classes of allocations in the instance. For each of these equivalence classes of allocations, we can check whether an allocation in the equivalence class is EF1 and has the required size vector in polynomial time, and output the equivalence class if so. Therefore, the overall running time is polynomial in m, as claimed. ◀

Lemma 9 implies that the decision problem can be solved efficiently for binary utilities.

▶ **Theorem 10.** *Reformability is in P for a constant number of agents with binary utilities.*

3.3 General Number of Agents

For any constant number of agents, the problem of determining the existence of an EF1 allocation with a given size vector is *weakly* NP-hard by Theorem 8 (even for identical utilities). For a general number of agents, the pseudopolynomial-time algorithm as described in Lemma 7 does not work, since that algorithm is at least exponential in the number of agents. Therefore, the decision problem for a general number of agents might not be *weakly* NP-hard. We show that the problem is indeed *strongly* NP-hard, even for identical utilities, by a reduction from 3-Partition, a strongly NP-hard problem [16, p. 224].

▶ **Theorem 11.** *Reformability is strongly NP-complete for identical utilities.*

The reduction in Theorem 11 requires us to check whether the set of agents holding the goods with utilities equal to the integers in the partition problem is EF1 towards the remaining agents. This stands in contrast to the reduction in Theorem 8, which entails checking whether the complementary set of agents is EF1 towards the remaining agents.

For binary utilities, the decision problem for a *constant* number of agents is in P (Theorem 10). The crucial reason is that in this case, the number of different types of goods is also a constant, which allows us to enumerate all the (essentially equivalent) EF1 allocations in polynomial time (Lemma 9). This is no longer possible when the number of agents is non-constant. In fact, we show that the decision problem is NP-hard for a general number of agents with binary utilities. To this end, we reduce from Graph k-Colorability, which is NP-hard for any fixed $k \geq 3$ [16, p. 191].

▶ **Theorem 12.** *Reformability is NP-complete for binary utilities.*

Proof. Clearly, this problem is in NP. Therefore, it suffices to show that it is NP-hard.

To this end, we shall reduce from Graph k-Colorability with $k \geq 3$. In Graph k-Colorability, we are given a graph $G = (V, E)$ and a positive integer k, and the problem is to decide whether G is k-colorable, i.e., whether each of the vertices in V can be assigned one of k colors in such a way that no two adjacent vertices are assigned the same color. This decision problem is known to be NP-hard for any fixed $k \geq 3$ [16, p. 191].

Let an instance of Graph k-Colorability be given with a fixed $k \geq 3$, where $V = \{v_1, \ldots, v_p\}$ and $E = \{e_1, \ldots, e_q\}$. Define a fair division instance as follows. There are $n = q + k$ agents where the first q agents are called *edge agents* and the last k agents are called *color agents*. There are $m = kp$ goods. Each color agent assigns zero utility to every good. For $r \in \{1, \ldots, q\}$, if $e_r = \{v_i, v_j\}$, then the r^{th} edge agent assigns a utility of 1 to each of g_i and g_j, and zero utility to every other good. Note that only the first p goods correspond to vertices and are valuable to the edge agents whose corresponding edges are incident to the vertices; the remaining $(k-1)p$ goods are not valuable to any agent. The size vector $\vec{s}$ is such that $s_r = 0$ for each edge agent r and $s_c = p$ for each color agent c. This reduction can be done in polynomial time. We claim that there exists an EF1 allocation with size vector $\vec{s}$ in this instance if and only if G is k-colorable.

($\Leftarrow$) Let a proper k-coloring of G be given. For $t \in \{1, \ldots, p\}$, if vertex v_t is assigned the color c, then allocate g_t to the color agent c. Since there are p such goods and each color agent is supposed to have p goods in her bundle, it is possible to allocate all of these goods. Subsequently, allocate the remaining goods arbitrarily among the color agents until every color agent has exactly p goods. We claim that this allocation is EF1. Every color agent assigns zero utility to every good and is thus EF1 towards every other agent. Each edge agent assigns a utility of 1 to only two goods, so we only need to check that these two goods are in different bundles. Indeed, these two goods correspond to vertices which are adjacent to each other in G, and proper coloring of G implies that the vertices are of different colors, so the corresponding goods are in different color agents' bundles. Hence, the allocation is EF1.

($\Rightarrow$) Let an EF1 allocation with size vector $\vec{s}$ be given. For $t \in \{1, \ldots, p\}$, if the good g_t is with color agent c, assign v_t to color c. We claim that this coloring is a proper k-coloring of G. Since there are k color agents, at most k colors are used. Therefore, it suffices to check that adjacent vertices are assigned different colors. Let $v_i, v_j \in V$ be adjacent vertices. Then, there exists an edge $e_r = \{v_i, v_j\}$. The edge agent r assigns a utility of 1 to each of g_i and g_j. Since agent r's bundle is empty, g_i and g_j must be in different (color agents') bundles in order for agent r to be EF1 towards every other agent. This implies that v_i and v_j are assigned different colors. ◀

Even though the decision problem is NP-hard for identical *or* binary utilities, we prove next that it can be solved in polynomial time for identical *and* binary utilities. Indeed, this can be done by checking whether the total number of valuable goods is within a certain threshold which can be computed in polynomial time. This threshold is in fact the sum over all agents $j \in N$ of the minimum between s_j and $1 + \min_{i \in N} s_i$. If the number of valuable goods is within this threshold, then we can distribute the valuable goods in a round-robin fashion first, thereby ensuring EF1. Conversely, if the number of valuable goods is beyond this threshold, then some agent i with the minimum s_i will not be EF1 towards another agent who receives at least $s_i + 2$ valuable goods.

▶ **Theorem 13.** *Reformability is in P for identical binary utilities.*

4 Optimal Number of Exchanges

In this section, we consider the complexity of computing the *optimal number of exchanges* required to reach an EF1 allocation from an initial allocation.

Recall that the decision problem in Section 3 is to determine whether there exists an EF1 allocation that can be reached from a given initial allocation. This is equivalent to determining whether the optimal number of exchanges to reach an EF1 allocation is finite or not. We have established a few scenarios where there exist polynomial-time algorithms for this task: (a) two agents with identical utilities (Theorem 5), (b) any constant number of agents with binary utilities (Theorem 10), and (c) any number of agents with identical binary utilities (Theorem 13). For these scenarios, we can run the respective polynomial-time algorithms to decide whether such an EF1 allocation exists – if none exists, then the optimal number of exchanges is ∞. Therefore, for the proofs in this section pertaining to these scenarios, we proceed with the assumption that such an EF1 allocation exists. We will show that the problem of computing the optimal number of exchanges for these scenarios is also in P; our algorithms can be modified to compute an optimal *sequence* of exchanges as well.

For the remaining scenarios where the decision problem in Section 3 is NP-hard, it is NP-hard to even decide whether the optimal number of exchanges to reach an EF1 allocation is finite or not. Therefore, for these scenarios, we shall focus on the special case where the

given size vector is *balanced*, so that the optimal number of exchanges is guaranteed to be finite (see Proposition 2). Even with this assumption, we will show that the computational problem for these scenarios remains NP-hard.

For convenience, we refer to as Optimal Exchanges the problem of deciding – given an instance, an initial allocation in the instance, and a number k – whether the optimal number of exchanges required to reach an EF1 allocation is at most k. Note that the optimal number of exchanges in the scenarios mentioned above are finite. By the proof of Proposition 1, the optimal number of exchanges in these scenarios is polynomial in the number of agents and the number of goods. As a result, Optimal Exchanges is in NP regardless of the number of agents, as we can easily verify whether an exchange path starting from the given initial allocation indeed reaches an EF1 allocation using at most k exchanges.

We begin with the case of two agents. For two agents with identical utilities, we show that there exists a polynomial-time algorithm that computes the optimal number of exchanges. This algorithm performs the exchanges until an EF1 allocation is reached, while keeping track of the number of exchanges required. The algorithm is "greedy" in the sense that at each step, it performs an exchange involving the most valuable good from the agent whose bundle has the higher utility, and the least valuable good from the other agent. We demonstrate that this choice is the best for the number of exchanges required to reach an EF1 allocation.

▶ **Theorem 14.** *Optimal Exchanges is in P for two agents with identical utilities.*

Proof. We show that we can compute the optimal number of exchanges in polynomial time. We assume that an EF1 allocation with the given size vector exists. If the initial allocation $\mathcal{A}$ is EF1, we are done. Otherwise, assume without loss of generality that agent 2 has a higher utility than agent 1 in $\mathcal{A}$. Let $\vec{s} = (s_1, s_2)$ be the size vector. By rearranging the labels of the goods, assume that the goods are in non-increasing order of utility, i.e., $u(g_1) \geq u(g_2) \geq \cdots \geq u(g_m)$. The algorithm proceeds as follows: repeatedly exchange the most valuable good in agent 2's bundle with the least valuable good in agent 1's bundle until agent 1 is EF1 towards agent 2. The optimal number of exchanges required is then the number of exchanges made in this algorithm.

First, we claim that in each exchange, a good from $\{g_1, \ldots, g_{s_1}\}$ in agent 2's bundle is always exchanged with a good from $\{g_{s_1+1}, \ldots, g_m\}$ in agent 1's bundle. Suppose on the contrary that this is not true. Let $\mathcal{A}'$ be the allocation just before we make the exchange that violates this claim. The only way for the claim to be violated is when $A'_1 = \{g_1, \ldots, g_{s_1}\}$ and $A'_2 = \{g_{s_1+1}, \ldots, g_m\}$. If $s_1 \leq s_2$, then by Lemma 4, there does not exist an EF1 allocation with size vector $\vec{s}$ – this would contradict our assumption that an EF1 allocation with size vector $\vec{s}$ exists. Otherwise, $s_1 > s_2$, and every good in A'_1 has a higher utility than every good in A'_2, so agent 1 is EF1 towards agent 2. This would contradict our assumption that the algorithm has not terminated. Hence, the claim is true.

By the claim in the previous paragraph, the total number of exchanges made is at most $\min\{s_1, s_2\} \leq m$. Each exchange can be performed in polynomial time, and so the algorithm terminates in polynomial time. We show next that an EF1 allocation is obtained when the algorithm terminates. It suffices to show that agent 2 is EF1 towards agent 1 in the final allocation. To this end, we show that agent 2 is EF1 towards agent 1 at every step of the algorithm. Let the initial allocation be $\mathcal{A}^0 = \mathcal{A}$, and let $\mathcal{A}^t$ be the allocation after t steps of the algorithm. Note that $\mathcal{A}^0$ satisfies the condition that agent 2 is EF1 towards agent 1, since agent 2 has a higher utility than agent 1 in $\mathcal{A}$. We show that if $\mathcal{A}^t$ has the property that agent 2 is EF1 towards agent 1 and agent 1 is *not* EF1 towards agent 2, then $\mathcal{A}^{t+1}$ has the

property that agent 2 is EF1 towards agent 1. Suppose that $g \in A_2^t$ is exchanged with $h \in A_1^t$; note that g is a good with the highest utility in A_2^t. This means that $u(A_1^t) < u(A_2^t \setminus \{g\})$. Then,

$$u(A_2^{t+1}) \geq u(A_2^{t+1} \setminus \{h\}) = u(A_2^t \setminus \{g\}) > u(A_1^t) \geq u(A_1^t \setminus \{h\}) = u(A_1^{t+1} \setminus \{g\}),$$

showing that agent 2 is EF1 towards agent 1 in $\mathcal{A}^{t+1}$.

Finally, we show that the optimal number of exchanges required to reach an EF1 allocation is at least the number of exchanges made in this algorithm. Let T be the number of exchanges made in this algorithm. For each $t \in \{1, \ldots, T\}$, let $g^t \in A_2$ (resp. $h^t \in A_1$) be the good in agent 2's bundle (resp. agent 1's bundle) that is exchanged at the t^{th} step of the algorithm. Note that $u(g^1) \geq \cdots \geq u(g^T) \geq u(h^T) \geq \cdots \geq u(h^1)$. Also, we have $u(A_1^{T-1}) < u(A_2^{T-1} \setminus \{g^T\})$, where g^T is a good with the highest utility in agent 2's bundle in $\mathcal{A}^{T-1}$. Suppose on the contrary that only $k \leq T-1$ exchanges are required to reach an EF1 allocation. Since $\mathcal{A}$ is not EF1, we have $1 \leq k < T$. Let (B_1, B_2) be the EF1 allocation after the k exchanges. The utility of B_1 is upper-bounded by the utility of A_1 after adding k goods of the highest utility from A_2 and removing k goods of the lowest utility from A_1, so

$$\begin{aligned}
u(B_1) &\leq u((A_1 \cup \{g^1, \ldots, g^k\}) \setminus \{h^1, \ldots, h^k\}) \\
&= u(A_1) + \sum_{t=1}^{k} (u(g^t) - u(h^t)) \\
&\leq u(A_1) + \sum_{t=1}^{T-1} (u(g^t) - u(h^t)) \\
&= u((A_1 \cup \{g^1, \ldots, g^{T-1}\}) \setminus \{h^1, \ldots, h^{T-1}\}) = u(A_1^{T-1}).
\end{aligned}$$

On the other hand, the utility of B_2 without the most valuable good is lower-bounded by the utility of A_2 after adding k goods of the lowest utility from A_1 and removing $k+1 \leq T$ goods of the highest utility from A_2, so we have

$$\begin{aligned}
u(B_2 \setminus \{g\}) &\geq u((A_2 \cup \{h^1, \ldots, h^k\}) \setminus \{g^1, \ldots, g^{k+1}\}) \\
&= u(A_2) - u(g^1) - \sum_{t=1}^{k} (u(g^{t+1}) - u(h^t)) \\
&\geq u(A_2) - u(g^1) - \sum_{t=1}^{T-1} (u(g^{t+1}) - u(h^t)) \\
&= u((A_2 \cup \{h^1, \ldots, h^{T-1}\}) \setminus \{g^1, \ldots, g^T\}) = u(A_2^{T-1} \setminus \{g^T\})
\end{aligned}$$

for every $g \in B_2$. This gives the inequality $u(B_1) \leq u(A_1^{T-1}) < u(A_2^{T-1} \setminus \{g^T\}) \leq u(B_2 \setminus \{g\})$ for all $g \in B_2$. Hence, agent 1 is not EF1 towards agent 2 in (B_1, B_2), contradicting the assumption that (B_1, B_2) is EF1. It follows that at least T exchanges are required to reach an EF1 allocation. ◀

However, if the utilities are not identical, then computing the optimal number of exchanges is NP-hard, even for balanced allocations. To show this hardness, we modify the construction from the proof of Theorem 6.

▶ **Theorem 15.** *Optimal Exchanges is NP-complete for two agents, even when the initial allocation is balanced.*

Proof. Clearly, this problem is in NP. To demonstrate NP-hardness, we modify the construction from the proof of Theorem 6. Recall that we have $Y = \{y_1, \dots, y_{2q+2}\}$ with $K < y_j \leq 2K$ for $j \in \{1, \dots, 2q\}$, $y_{2q+1} = 2K$, and $y_{2q+2} = 0$. A fair division instance $\mathcal{I}'$ is defined with $n = 2$ agents and a set of goods $M' = \{g_1, \dots, g_{2q+6}\}$. Agent 2's utility is such that $u_2(g_j) = y_j$ for $j \in \{1, \dots, 2q+2\}$, $u_2(g_{2q+3}) = u_2(g_{2q+4}) = 0$, and $u_2(g_{2q+5}) = u_2(g_{2q+6}) = 2K$. Agent 1's utility is such that $u_1(g) = u_2(g) + 4K$ for all $g \in M$. The size vector $\vec{s}'$ is $(q+2, q+4)$. In Theorem 6, it was proven that there exists an EF1 allocation with size vector $\vec{s}'$ in this instance if and only if Y can be partitioned into two multisets Y_1 and Y_2 of equal cardinalities (i.e., of size $q+1$ each) with sums $(q+3)K$ and $(q+1)K$ respectively. Both problems were proven to be NP-hard.

Define a new fair division instance $\mathcal{I}$ as follows. There are $n = 2$ agents and a set of goods $M = \{g_1, \dots, g_{4q+12}\}$. For $j \in \{1, \dots, 2q+6\}$, the utility of g_j for each agent is identical to that in the original fair division instance $\mathcal{I}'$. For $j \in \{2q+7, \dots, 4q+12\}$, we have $u_i(g_j) = 0$ for $i \in \{1, 2\}$. The size vector $\vec{s}$ is $(2q+6, 2q+6)$. In the initial allocation $\mathcal{A}$, agent 1 has $A_1 = \{g_{2q+7}, \dots, g_{4q+12}\}$ and agent 2 has $A_2 = \{g_1, \dots, g_{2q+6}\}$. This reduction can be done in polynomial time. We claim that the optimal number of exchanges required to reach an EF1 allocation from $\mathcal{A}$ is at most $q+2$ in $\mathcal{I}$ if and only if there exists an EF1 allocation with size vector $\vec{s}'$ in $\mathcal{I}'$.

($\Leftarrow$) Let (A'_1, A'_2) be an EF1 allocation with size vector $\vec{s}'$ in $\mathcal{I}'$. Note that $A'_1 \subseteq A_2$ and $|A'_1| = q+2$. In $\mathcal{A}$, exchange the $q+2$ goods from A'_1 with any $q+2$ goods in A_1. This requires a total of $q+2$ exchanges. The new allocation has exactly the same goods as that in (A'_1, A'_2) along with other goods with zero utility, and so is EF1. Therefore, the optimal number of exchanges to reach an EF1 allocation from $\mathcal{A}$ is at most $q+2$.

($\Rightarrow$) Suppose that an EF1 allocation $\mathcal{B}$ is reached from $\mathcal{A}$ after $t \leq q+2$ exchanges in $\mathcal{I}$. We may assume that every good is not exchanged more than once. By the same reasoning as in the proof of Theorem 6, we must have $u_1(B_1) \geq (5q+13)K$ and $u_2(B_1) \leq (q+5)K$. Since t goods are transferred from A_2, we have $u_1(B_1) = u_2(B_1) + t(4K) \leq (q+5)K + (q+2)(4K) = (5q+13)K$. This means that the inequalities for $u_1(B_1)$ are tight, and we have $u_1(B_1) = 5q+13$ and $t = q+2$. Letting $M_1 = A_2 \cap B_1$, we have $|M_1| = q+2$ and $M_1 \subseteq A_2 \subseteq M'$. Since (B_1, B_2) is an EF1 allocation, the allocation that removes all goods with zero utility is also EF1, namely, $(M_1, M' \setminus M_1)$. This induces an EF1 allocation with size vector $\vec{s}'$ in $\mathcal{I}'$. ◀

While a polynomial-time algorithm to compute the optimal number of exchanges exists for two agents with identical utilities, this is not the case for three or more agents unless P = NP. Indeed, we establish the NP-hardness of this problem via a reduction from Balanced Multi-Partition with $p \geq 2$, an NP-hard problem by Proposition 3.

▶ **Theorem 16.** *Optimal Exchanges is NP-complete for $n \geq 3$ agents with identical utilities, where n is a constant, even when the initial allocation is balanced.*

Next, we consider binary utilities. We have shown that deciding whether the optimal number of exchanges to reach an EF1 allocation is finite can be done in polynomial time (Theorem 10). We now establish that the same is true for computing this exact number.

▶ **Theorem 17.** *Optimal Exchanges is in P for a constant number of agents with binary utilities.*

Let us now consider a non-constant number of agents. We have shown that computing the optimal number of exchanges required to reach an EF1 allocation is NP-hard, even for identical utilities (Theorem 16). We thus consider binary utilities.

Although this problem belongs to P when the number of agents is a constant (Theorem 17), we show that it is NP-hard for a general number of agents, even for the special case where the initial allocation is balanced. To this end, we reduce from Exact Cover by 3-Sets.

▶ **Theorem 18.** *Optimal Exchanges is NP-complete for binary utilities, even when the initial allocation is balanced.*

Finally, we consider identical binary utilities. We show that for this class of utilities, the computational problem can be solved efficiently regardless of whether the size vector of the initial allocation is balanced or not. To show this, we demonstrate that the greedy algorithm allows an EF1 allocation to be reached using the smallest number of exchanges.

▶ **Theorem 19.** *Optimal Exchanges is in P for identical binary utilities.*

5 Worst-Case Bounds

In this section, instead of instance-specific optimization, we turn our attention to the *worst-case* number of exchanges required to reach an EF1 allocation from some initial allocation. Since an EF1 allocation may not always be reachable (as can be seen from Section 3), we shall focus on the special case where the number of goods in each agent's bundle is the same. We say that a size vector $\vec{s} = (s_1, \ldots, s_n)$ is *s-balanced* for a positive integer s if $s_i = s$ for all $i \in N$, and an allocation is *s-balanced* if it has an s-balanced size vector. We shall consider the worst-case number of exchanges starting from an s-balanced allocation for n agents.

Given n and s, let $f(n, s)$ be the smallest integer such that for every instance with n agents and ns goods and every s-balanced allocation $\mathcal{A}$ in the instance, there exists an EF1 allocation that can be reached from $\mathcal{A}$ using at most $f(n, s)$ exchanges. We shall examine the bounds for $f(n, s)$.

We first derive an upper bound for $f(n, s)$. At a high level, we use an algorithm by Biswas and Barman [4] to find an EF1 allocation under cardinality constraints such that every agent retains roughly s/n of her goods from her original bundle. The algorithm also distributes the goods in each agent's initial bundle to the other agents as evenly as possible in order to maximize the number of goods that can be exchanged one-to-one, thereby minimizing the total number of exchanges. One can check that roughly $s(n-1)/2$ exchanges are required to reach this EF1 allocation from the initial allocation.

▶ **Theorem 20.** *Let n and s be positive integers, and let $q = \lfloor s/n \rfloor$ and $r = s - qn$ be the quotient and remainder when s is divided by n respectively. Then,*

$$f(n, s) \leq \begin{cases} s(n-1)/2 & \text{if } r = 0; \\ s(n-1)/2 + r(n-3)/2 + 1 & \text{otherwise.} \end{cases}$$

Moreover, we have $f(2, s) \leq (s - r)/2$ for all s.

Proof. Let $\mathcal{A}$ be an s-balanced allocation. It suffices to find an EF1 s-balanced allocation $\mathcal{B}$ such that the optimal number of exchanges to reach $\mathcal{B}$ from $\mathcal{A}$ is at most the expression given in the theorem statement.

When $n = 2$, allocate the goods in A_1 to the two agents in a round-robin fashion with agent 1 going first, and allocate the goods in A_2 to the two agents in a round-robin fashion with agent 2 going first. Call this new allocation $\mathcal{B}$. Note that $\mathcal{B}$ is clearly s-balanced. We have $A_i \cap B_{3-i} = (s - r)/2$ for $i \in \{1, 2\}$, so the optimal number of exchanges required to reach $\mathcal{B}$ from $\mathcal{A}$ is (exactly) $(s - r)/2$. To see that $\mathcal{B}$ is EF1, observe that agent 1 does not envy agent 2 with respect to the goods chosen from A_1 and is EF1 towards agent 2 with respect to the goods chosen from A_2, so agent 1 is EF1 towards agent 2 in $\mathcal{B}$; likewise, agent 2 is EF1 towards agent 1 in $\mathcal{B}$. This shows that $f(2, s) \leq (s - r)/2$.

When $n \geq 3$, we shall find an EF1 s-balanced allocation $\mathcal{B}$ by generalizing the method for two agents. We define $n + r$ categories of goods $C_1, \ldots, C_n, D_1, \ldots, D_r$ as follows. For $i \in N$, category C_i contains qn goods arbitrarily selected from A_i only; note that r goods remain unselected in A_i. Next, we form D_w recursively as follows: let $w \in \{1, \ldots, r\}$ be the smallest index such that D_w does not have n goods yet, let $i \in N$ be the smallest index such that A_i still has unselected goods, arbitrarily select a good in A_i, and add it to D_w. At the end of this process, every category C_i has exactly qn goods from A_i, and every category D_w has exactly n goods from consecutive agents' bundles, say, $A_{i_w}, A_{i_w+1}, \ldots, A_{j_w}$.

We now proceed to form $\mathcal{B}$ using the algorithm by Biswas and Barman [4] which finds an EF1 allocation under cardinality constraints. In particular, there exists an EF1 allocation $\mathcal{B} = (B_1, \ldots, B_n)$ such that $|C_i \cap B_j| = |C_i|/n = q$ for all $i, j \in N$ and $|D_w \cap B_j| = |D_w|/n = 1$ for all $w \in \{1, \ldots, r\}$, $j \in N$. Also, $\mathcal{B}$ is s-balanced because $|B_j| = qn + r = s$ for all $j \in N$. We shall bound the number of exchanges required to reach $\mathcal{B}$ from $\mathcal{A}$.

For each unordered pair of distinct $i, j \in N$, exchange the q goods from $C_i \cap B_j$ (which are in A_i) with the q goods from $C_j \cap B_i$ (which are in A_j). This requires a total of $qn(n-1)/2$ exchanges. Call this intermediate allocation $\mathcal{A}' = (A'_1, \ldots, A'_n)$. At this point, the only goods that are possibly in the wrong bundles in $\mathcal{A}'$ (as compared to $\mathcal{B}$) are the goods in all the D_w, and there are at most rn such goods. For each $i \in N$, let $X_i = A'_i \cap (D_1 \cup \cdots \cup D_r)$.

If $r = 0$, then $\mathcal{A}' = \mathcal{B}$, and we are done since the total number of exchanges is $qn(n-1)/2 = s(n-1)/2$. Else, $r > 0$. Consider the directed graph where the vertices are the agents and each edge e_g represents a good $g \in M$ such that if $g \in A'_i \cap B_j$, then $e_g = (i, j)$. Igarashi et al. [22, Prop. 4.1] showed that the number of exchanges required to reach $\mathcal{B}$ from $\mathcal{A}'$ is $m - c^*$, where c^* is the maximum possible cardinality of a partition of the edges of the graph into (directed) circuits. In $\mathcal{A}'$, qn^2 goods from all the C_i are in the correct bundle by the previous process, and each of the edges representing these goods has its own circuit, say, (i, i) if the good is in A'_i. We shall show that the edges representing the rn goods in all the D_w can be partitioned into at least $2r - 1$ disjoint circuits. This will give at least $qn^2 + (2r - 1) = sn - (rn - 2r + 1)$ as the cardinality of one such partition of the edges of the graph into circuits. Accordingly, $c^* \geq sn - (rn - 2r + 1)$, and the number of exchanges required to reach $\mathcal{B}$ from $\mathcal{A}'$ is $m - c^* \leq rn - 2r + 1$. Then, the number of exchanges required to reach $\mathcal{B}$ from $\mathcal{A}$ (via $\mathcal{A}'$) is at most $qn(n-1)/2 + (rn - 2r + 1) = s(n-1)/2 + r(n-3)/2 + 1$, establishing the theorem.

Let $w \in \{1, \ldots, r\}$ be given. We shall show that there exists a cycle formed with a subset of the edges representing the goods in D_w. The goods in D_w come from consecutive agents' bundles in $\mathcal{A}'$, say, agents i_w to j_w. Every agent receives exactly one good from D_w in $\mathcal{B}$; in particular, agents i_w to j_w receive exactly one good from D_w each. Consider the good g in $D_w \cap B_{i_w}$. If g is in X_{i_w}, then the edge $e_g = (i_w, i_w)$ is a desired cycle. Otherwise, g belongs to some agent $i' \in \{i_w + 1, \ldots, j_w\}$ in $\mathcal{A}'$. Then, the edge e_g is (i', i_w). Next, we consider the good g' in $D_w \cap B_{i'}$, and find the agent that has g' in $\mathcal{A}'$. The edge representing g' then points to i' from that agent. By repeating this, we eventually find a cycle formed with some of these edges and with a subset of the agents i_w to j_w as vertices. Let $M_w \subseteq D_w$ be the set of goods that are represented by the edges in this cycle. Note that each X_i for $i \in \{i_w, \ldots, j_w\}$ contains at most one good in M_w, and each X_i for $i \in N \setminus \{i_w, \ldots, j_w\}$ does not contain any good in M_w.

Now, consider the goods represented by the edges of the r cycles – one for each w. Note that these cycles are disjoint since the sets M_w are pairwise disjoint. Let $M_0 = \bigcup_{w=1}^{r} M_w$. We claim that $|M_0| < 2n$. Since the r goods in X_1 are entirely contained in D_1, we have $|X_1 \cap M_1| \leq 1$ and $|X_1 \cap M_w| = 0$ for $w \in \{2, \ldots, r\}$, which implies that $|\bigcup_{w=1}^{r} (X_1 \cap M_w)| \leq 1$.

Now, for each $i \in N \setminus \{1\}$, the r goods in X_i can only be contained in at most two D_w – to see this, observe that if the r goods are contained in $D_{w'}$, $D_{w'+1}$, and $D_{w'+2}$, then $D_{w'+1} \subseteq X_i$, which implies that $r = |X_i| \geq |D_{w'+1}| = n$, a contradiction. Thus, we have $|X_i \cap M_w| \leq 1$ for all $w \in \{1, \ldots, r\}$, and $|X_i \cap M_w| = 1$ for at most two w, and so $|\bigcup_{w=1}^{r}(X_i \cap M_w)| \leq 2$. Since $M_0 = \bigcup_{i \in N} \bigcup_{w=1}^{r}(X_i \cap M_w)$, we have $|M_0| \leq 1 + (n-1) \cdot 2 < 2n$, proving the claim.

Finally, consider the edges representing the rn goods in all the D_w. We have shown that fewer than $2n$ of these edges can be used to form r disjoint circuits (in fact, cycles). There are more than $rn - 2n = (r-2)n$ edges remaining. Since we can always require every circuit to have length at most n, there exists a partition of the remaining edges into more than $r-2$ disjoint circuits, i.e., at least $r-1$ disjoint circuits. The total number of circuits in this partition is at least $r + (r-1) = 2r - 1$. This completes the proof. ◀

If no good is involved in more than one exchange, then $s(n-1)/2$ exchanges means that a total of $s(n-1) = m(1-1/n)$ goods are exchanged. When n is large, the fraction of goods involved in the exchanges becomes close to 1. While this bound might not seem impressive, we show next that it is, in fact, already essentially tight. Specifically, we establish a lower bound for $f(n, s)$ by constructing an instance (with binary utilities) and an s-balanced allocation $\mathcal{A}$ in the instance such that roughly $s(n-1)/2$ exchanges are necessary to reach an EF1 allocation from $\mathcal{A}$.

▶ **Theorem 21.** *Let n and s be positive integers, and let $q = \lfloor s/n \rfloor$ and $r = s - qn$ be the quotient and remainder when s is divided by n respectively. Then,*

$$f(n,s) \geq \begin{cases} s(n-1)/2 & \text{if } r = 0; \\ s(n-1)/2 - (n-r)/2 & \text{otherwise.} \end{cases}$$

For two agents, Theorems 20 and 21 give a tight bound of $f(2, s) = (s-r)/2 = m/4 - r/2 = \lfloor m/4 \rfloor$. This means that in the worst-case scenario, the number of exchanges required to reach an EF1 allocation is roughly one-quarter of the total number of goods between the two agents, or equivalently, roughly half of the goods need to be exchanged between the two agents to reach an EF1 allocation.

Theorems 20 and 21 also give a tight bound of $f(n, s) = s(n-1)/2$ whenever s is divisible by n. By observing the proof of Theorem 20, we can achieve an EF1 allocation with $f(n, s)$ exchanges without involving each good in more than one exchange. This means that a $(1 - 1/n)$ fraction of all goods need to be exchanged in the worst-case scenario. Intuitively, this happens when each agent only values the goods in the bundle of every agent except her own in the initial allocation, and therefore needs to ensure that these goods are evenly distributed among all agents including herself.

Define $f_{\text{bin}}(n, s)$ as the smallest integer such that for every *binary* instance with n agents and ns goods and every s-balanced allocation $\mathcal{A}$ in the instance, there exists an EF1 allocation that can be reached from $\mathcal{A}$ using at most $f_{\text{bin}}(n, s)$ exchanges. The proof of Theorem 21 uses a binary instance, which means that the lower bound of the theorem works for f_{bin} as well. Clearly, the upper bound of Theorem 20 works for f_{bin}, so the discussion in the preceding paragraphs also applies to binary instances too.

Given n and s, let $f_{\text{id,bin}}(n, s)$ be the smallest integer such that for every instance with n agents with *identical binary* utilities and ns goods and every s-balanced allocation $\mathcal{A}$ in the instance, there exists an EF1 allocation that can be reached from $\mathcal{A}$ using at most $f_{\text{id,bin}}(n, s)$ exchanges. We show that $f_{\text{id,bin}}(n, s)$ is roughly $sn/4$ for even n and $s(n-1)(n+1)/4n$ for odd n – note that this is approximately half of the bound $f(n, s)$ for arbitrary utilities, which

is roughly $s(n-1)/2$ as seen earlier. The upper bounds (of $sn/4$ and $s(n-1)(n+1)/4n$ respectively) correspond to the case where half of the agents have all the valuable goods while the remaining half have all the non-valuable goods.

▶ **Theorem 22.** *Let n and s be positive integers. If n is even, then $\frac{n}{2}\left\lfloor\frac{s}{2}\right\rfloor \leq f_{\mathrm{id,bin}}(n,s) \leq \frac{sn}{4}$. If n is odd, then $\frac{n+1}{2}\left\lfloor\frac{s(n-1)}{2n}\right\rfloor \leq f_{\mathrm{id,bin}}(n,s) \leq \frac{s(n-1)(n+1)}{4n}$.*

6 Conclusion and Future Work

We have studied the reformability of unfair allocations and the number of exchanges required in the reformation process. We uncovered several distinctions in the complexity of these problems based on the number of agents and their utility functions, and showed that the number of exchanges required to reach an EF1 allocation is relatively high in the worst case.

While our worst-case bounds for general utilities are already exactly tight in certain scenarios and almost tight generally, an open question is to tighten them for more than two agents when the number of goods in each agent's bundle is not divisible by the number of agents. Another interesting direction is to require each exchange to be beneficial for both agents involved. One could also consider the model of *transferring* goods instead of *exchanging* them – an EF1 allocation is always reachable from any allocation in this model, so a natural question is to determine the optimal number of exchanges needed for this goal.

References

1 Haris Aziz, Péter Biró, Jérôme Lang, Julien Lesca, and Jérôme Monnot. Efficient reallocation under additive and responsive preferences. *Theoretical Computer Science*, 790:1–15, 2019. doi:10.1016/J.TCS.2019.05.011.

2 Haris Aziz, Ildikó Schlotter, and Toby Walsh. Control of fair division. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 67–73, 2016. URL: http://www.ijcai.org/Abstract/16/017.

3 Matthias Bentert, Robert Bredereck, Eva Deltl, Pallavi Jain, and Leon Kellerhals. How to resolve envy by adding goods. In *Proceedings of the 34th International Joint Conference on Artificial Intelligence (IJCAI)*, 2025.

4 Arpita Biswas and Siddharth Barman. Fair division under cardinality constraints. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 91–97, 2018. doi:10.24963/IJCAI.2018/13.

5 Niclas Boehmer, Robert Bredereck, Klaus Heeger, Dušan Knop, and Junjie Luo. Multivariate algorithmics for eliminating envy by donating goods. *Autonomous Agents and Multi-Agent Systems*, 38(2):43:1–43:35, 2024.

6 Steven J. Brams and Alan D. Taylor. *Fair Division: From Cake-Cutting to Dispute Resolution*. Cambridge University Press, 1996.

7 Felix Brandt and Anaëlle Wilczynski. On the convergence of swap dynamics to Pareto-optimal matchings. *Journal of Artificial Intelligence Research*, 80:1063–1098, 2024. doi:10.1613/JAIR.1.15305.

8 Robert Bredereck, Andrzej Kaczmarczyk, Junjie Luo, Rolf Niedermeier, and Florian Sachse. Improving resource allocations by sharing in pairs. *Journal of Artificial Intelligence Research*, 78:1069–1109, 2023. doi:10.1613/JAIR.1.15001.

9 Eric Budish, Gérard P. Cachon, Judd B. Kessler, and Abraham Othman. Course Match: a large-scale implementation of approximate competitive equilibrium from equal incomes for combinatorial allocation. *Operations Research*, 65(2):314–336, 2017. doi:10.1287/OPRE.2016.1544.

10 Harish Chandramouleeswaran, Prajakta Nimbhorkar, and Nidhi Rathi. Fair division in a variable setting. In *Proceedings of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 472–480, 2025. doi:10.5555/3709347.3743562.

11 Jiehua Chen, Joanna Kaczmarek, Paul Nüsken, Jörg Rothe, Ildikó Schlotter, and Tessa Seeger. Control in computational social choice. In *Proceedings of the 34th International Joint Conference on Artificial Intelligence (IJCAI)*, 2025.

12 Yann Chevaleyre, Ulle Endriss, Sylvia Estivie, and Nicolas Maudet. Reaching envy-free states in distributed negotiation settings. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence (IJCAI)*, pages 1239–1244, 2007. URL: http://ijcai.org/Proceedings/07/Papers/200.pdf.

13 Anastasia Damamme, Aurélie Beynier, Yann Chevaleyre, and Nicolas Maudet. The power of swap deals in distributed resource allocation. In *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 625–633, 2015. URL: http://dl.acm.org/citation.cfm?id=2773235.

14 Britta Dorn, Ronald de Haan, and Ildikó Schlotter. Obtaining a proportional allocation by deleting items. *Algorithmica*, 83(5):1559–1603, 2021. doi:10.1007/S00453-020-00794-4.

15 Piotr Faliszewski and Jörg Rothe. Control and bribery in voting. In Felix Brandt, Vincent Conitzer, Ulle Endriss, Jérôme Lang, and Ariel D. Procaccia, editors, *Handbook of Computational Social Choice*, chapter 7, pages 146–168. Cambridge University Press, 2016. doi:10.1017/CBO9781107446984.008.

16 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

17 Jonathan Goldman and Ariel D. Procaccia. Spliddit: Unleashing fair division algorithms. *ACM SIGecom Exchanges*, 13(2):41–46, 2014. doi:10.1145/2728732.2728738.

18 Laurent Gourvès, Julien Lesca, and Anaëlle Wilczynski. Object allocation via swaps along a social network. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 213–219, 2017. doi:10.24963/IJCAI.2017/31.

19 Jiatong Han and Warut Suksompong. Fast & Fair: a collaborative platform for fair division applications. In *Proceedings of the 38th AAAI Conference on Artificial Intelligence (AAAI)*, pages 23796–23798, 2024. doi:10.1609/AAAI.V38I21.30568.

20 Sen Huang and Mingyu Xiao. Object reachability via swaps under strict and weak preferences. *Autonomous Agents and Multi-Agent Systems*, 34(2):51:1–51:33, 2020.

21 Halvard Hummel and Magnus Lie Hetland. Maximin shares under cardinality constraints. In *Proceedings of the 19th European Conference on Multi-Agent Systems (EUMAS)*, pages 188–206, 2022. doi:10.1007/978-3-031-20614-6_11.

22 Ayumi Igarashi, Naoyuki Kamiyama, Warut Suksompong, and Sheung Man Yuen. Reachability of fair allocations via sequential exchanges. *Algorithmica*, 86(12):3653–3683, 2024. doi:10.1007/S00453-024-01271-Y.

23 Ayumi Igarashi and Tomohiko Yokoyama. Kajibuntan: a house chore division app. In *Proceedings of the 37th AAAI Conference on Artificial Intelligence (AAAI)*, pages 16449–16451, 2023. doi:10.1609/AAAI.V37I13.27075.

24 Takehiro Ito, Yuni Iwamasa, Naonori Kakimura, Naoyuki Kamiyama, Yusuke Kobayashi, Yuta Nozaki, Yoshio Okamoto, and Kenta Ozeki. Reforming an envy-free matching. *Algorithmica*, 87(4):594–620, 2025. doi:10.1007/S00453-025-01294-Z.

25 Takehiro Ito, Naonori Kakimura, Naoyuki Kamiyama, Yusuke Kobayashi, Yuta Nozaki, Yoshio Okamoto, and Kenta Ozeki. On reachable assignments under dichotomous preferences. *Theoretical Computer Science*, 979:114196, 2023. doi:10.1016/J.TCS.2023.114196.

26 Fu Li, C. Gregory Plaxton, and Vaibhav B. Sinha. Object allocation over a network of objects: Mobile agents with strict preferences. In *Proceedings of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1578–1580, 2021. doi:10.5555/3463952.3464165.

27 Hervé Moulin. *Fair Division and Collective Welfare*. MIT Press, 2003.

28 Hervé Moulin. Fair division in the internet age. *Annual Review of Economics*, 11:407–441, 2019.

29 Luis Müller and Matthias Bentert. On reachable assignments in cycles. In *Proceedings of the 7th International Conference on Algorithmic Decision Theory (ADT)*, pages 273–288, 2021. doi:10.1007/978-3-030-87756-9_18.

30 Jack Robertson and William Webb. *Cake-Cutting Algorithms: Be Fair if You Can*. Peters/CRC Press, 1998.

31 Erel Segal-Halevi. Redividing the cake. *Autonomous Agents and Multi-Agent Systems*, 36(1):14:1–14:36, 2022.

32 Hila Shoshan, Noam Hazon, and Erel Segal-Halevi. Efficient nearly-fair division with capacity constraints. In *Proceedings of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 206–214, 2023. doi:10.5555/3545946.3598638.

33 Xiaowei Wu, Bo Li, and Jiarui Gan. Approximate envy-freeness in indivisible resource allocation with budget constraints. *Information and Computation*, 303:105264, 2025. doi:10.1016/J.IC.2024.105264.

34 Sheung Man Yuen, Ayumi Igarashi, Naoyuki Kamiyama, and Warut Suksompong. Reforming an unfair allocation by exchanging goods. *CoRR*, abs/2412.19264, 2024. doi:10.48550/arXiv.2412.19264.