20th International Symposium on Parameterized and Exact Computation

IPEC 2025, September 17–19, 2025, Warsaw, Poland

Edited by

Akanksha Agrawal

Erik Jan van Leeuwen

Editors

Akanksha Agrawal
Indian Institute of Technology Madras, Chennai, India
akanksha@cse.iitm.ac.in

Erik Jan van Leeuwen
Utrecht University, The Netherlands
e.j.vanleeuwen@uu.nl

ACM Classification 2012
Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Algorithm design techniques; Theory of computation → Graph algorithms analysis; Theory of computation → Mathematical optimization; Theory of computation → Complexity classes; Theory of computation → Approximation algorithms analysis; Mathematics of computing → Discrete mathematics

ISBN 978-3-95977-407-9

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/978-3-95977-407-9.

Publication date
December, 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.IPEC.2025.0

LIPIcs – Leibniz International Proceedings in Informatics

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

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
Akanksha Agrawal and Erik Jan van Leeuwen 0:ix

The Tradition of I(W)PEC
..... 0:xi

Committees
..... 0:xiii

List of Authors
..... 0:xv

Invited Talk

A Brief History of Parameterized Algorithms for Block-Structured Integer Programs
Martin Koutecký 1:1–1:20

Regular Papers

Directed Disjoint Paths Remains W[1]-Hard on Acyclic Digraphs Without Large Grid Minors
Ken-ichi Kawarabayashi, Nicola Lorenz, Marcelo Garlet Milani, and Jacob Stegemann 2:1–2:17

Parameterized Maximum Node-Disjoint Paths
Michael Lampis and Manolis Vasilakis 3:1–3:15

On the Complexity of Secluded Path Problems
Tesshu Hanaka and Daisuke Tsuru 4:1–4:16

Kernelization for H-Coloring
Yael Berkman and Ishay Haviv 5:1–5:17

Boundaried Kernelization via Representative Sets
Leonid Antipov and Stefan Kratsch 6:1–6:13

Parameterized Complexity of Scheduling Unit-Time Jobs with Generalized Precedence Constraints
Christina Büsing, Maurice Draeger, and Corinna Mathwieser 7:1–7:16

Binary k-Center with Missing Entries: Structure Leads to Tractability
Tobias Friedrich, Kirill Simonov, and Farehe Soheil 8:1–8:19

An ETH-Tight FPT Algorithm for Rejection-Proof Set Packing with Applications to Kidney Exchange
Bart M. P. Jansen, Jeroen S. K. Lamme, and Ruben F. A. Verhaegh 9:1–9:15

Parameterized Complexity of Vehicle Routing
Michelle Döring, Jan Fehse, Tobias Friedrich, Paula Marten, Niklas Mohrin, Kirill Simonov, Farehe Soheil, Jakob Timm, and Shaily Verma 10:1–10:17

20th International Symposium on Parameterized and Exact Computation (IPEC 2025).
Editors: Akanksha Agrawal and Erik Jan van Leeuwen
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Parameterized Algorithms for Diversity of Networks with Ecological Dependencies
Mark Jones and Jannik Schestag 11:1–11:21

Timeline Problems in Temporal Graphs: Vertex Cover vs. Dominating Set
Anton Herrmann, Christian Komusiewicz, Nils Morawietz, and Frank Sommer ... 12:1–12:18

On Maximum 2-Clubs
Joanne Dumont, Michael Lampis, Mathieu Liedloff, Anthony Perez, and Ioan Todinca 13:1–13:14

A Simple Algorithm for Combinatorial n-Fold ILPs Using the Steinitz Lemma
Sushmita Gupta, Pallavi Jain, Sanjay Seetharaman, and Meirav Zehavi 14:1–14:17

New Algorithm for Combinatorial n-Folds and Applications
Klaus Jansen, Kai Kahler, Lis Pirotton, and Malte Tutas 15:1–15:15

Designing Compact ILPs via Fast Witness Verification
Michał Włodarczyk 16:1–16:18

Treedepth Inapproximability and Exponential ETH Lower Bound
Édouard Bonnet, Daniel Neuen, and Marek Sokołowski 17:1–17:10

Bridging Treewidth and Clique-Width via Cograph-Modular-Treewidth
Václav Blažej, Satyabrata Jana, M. S. Ramanujan, and Peter Strulo 18:1–18:18

Tight Bounds for Connected Odd Cycle Transversal Parameterized by Clique-Width
Narek Bojikian and Stefan Kratsch 19:1–19:15

Deterministically Counting k-Paths and Trees Parameterized by Treewidth in Single-Exponential Time
Jonne Visser and Hans L. Bodlaender 20:1–20:14

A Polynomial Delay Algorithm Generating All Potential Maximal Cliques in Triconnected Planar Graphs
Alexander Grigoriev, Yasuaki Kobayashi, Hisao Tamaki, and Tom C. van der Zanden 21:1–21:17

Efficient Enumeration of k-Plexes and k-Defective Cliques
Mohamed Jiddou and George Manoussakis 22:1–22:16

Enumeration Kernels for Vertex Cover and Feedback Vertex Set
Marin Bougeret, Guilherme C. M. Gomes, Vinicius F. dos Santos, and Ignasi Sau 23:1–23:18

A Note on the Parameterised Complexity of Coverability in Vector Addition Systems
Michał Pilipczuk, Sylvain Schmitz, and Henry Sinclair-Banks 24:1–24:19

Exact Algorithms and Hardness Result for the Boolean Connectivity Problem of k-Horn Formulas
Takashi Horiyama, Yuto Okura, Kazuhisa Seto, and Junichi Teruyama 25:1–25:15

Complexity of Local Search for CSPs Parameterized by Constraint Difference
Aditya Anand, Vincent Cohen-Addad, Tommaso D'Orsi, Anupam Gupta, Euiwoong Lee, Debmalya Panigrahi, and Sijin Peng 26:1–26:17

Uniformity Within Parameterized Circuit Classes
Steef Hegeman, Jan Martens, and Alfons Laarman 27:1–27:16

Geodetic Set on Graphs of Constant Pathwidth and Feedback Vertex Set Number
Prafullkumar Tale 28:1–28:14

Hitting Geodesic Intervals in Structurally Restricted Graphs
Tatsuya Gima, Yasuaki Kobayashi, Yuto Okada, Yota Otachi, and Hayato Takaike 29:1–29:16

A Graph Width Perspective on Partially Ordered Hamiltonian Paths and Cycles II: Vertex and Edge Deletion Numbers
Jesse Beisegel, Katharina Klost, Kristin Knorr, Fabienne Ratajczak, and Robert Scheffler 30:1–30:19

Hamiltonicity Parameterized by Mim-Width Is (Indeed) Para-NP-Hard
Benjamin Bergougnoux and Lars Jaffke 31:1–31:10

The PACE 2025 Parameterized Algorithms and Computational Experiments Challenge

The PACE 2025 Parameterized Algorithms and Computational Experiments Challenge: Dominating Set and Hitting Set
Mario Grobler and Sebastian Siebertz 32:1–32:17

PACE Solver Description: OBLX Exact Solver for the Dominating Set Problem
Jona Dirks, Enna Gerhard, Victoria Kaial, and Lucas Lorieau 33:1–33:4

PACE Solver Description: Shadoks Approach to Minimum Hitting Set and Dominating Set
Guilherme D. da Fonseca, Fabien Feschet, and Yan Gerard 34:1–34:5

PACE Solver Description: Bad Dominating Set Maker
Alexander Dobler, Simon Dominik Fink, and Mathis Rocton 35:1–35:5

PACE Solver Description: Reductions and Heuristic Search for the Dominating Set Problem and the Hitting Set Problem
Florian Fontan and Guillaume Verger 36:1–36:3

PACE Solver Description: Minimum Hitting Set Computation via Core-Guided MaxSAT Solving
André Schidler 37:1–37:4

PACE Solver Description: HitS&DoSeS – Exact and Heuristic Solvers for the Dominating Set and Hitting Set Problems
Sylwester Swat 38:1–38:4

PACE Solver Description: UzL Solver for Dominating Set and Hitting Set
Max Bannach, Florian Chudigiewitsch, and Marcel Wienöbst 39:1–39:4

PACE Solver Description: Weighting-Based Local Search Heuristic for the Hitting Set Problem
Canhui Luo, Qingyun Zhang, Zhouxing Su, and Zhipeng Lü 40:1–40:4

Preface

The International Symposium on Parameterized and Exact Computation (IPEC) is an annual conference covering all aspects of parameterized and exact algorithms and complexity. It started in 2004 as a biennial workshop and has been an annual conference since 2009 (see the tradition of I(W)PEC on the next page). Its 20th edition took place at the University of Warsaw in Warsaw, Poland, on 17-19 September, 2025, as part of ALGO 2025.

This volume contains the papers presented at IPEC 2025. In all, 64 extended abstracts were registered for the conference, out of which 3 were withdrawn, and thus 61 full submissions were finally received. This represents an increase compared to 2024, when 51 full submissions were received. The program committee selected 30 out of the 61 submissions for presentation at the conference. These 30 papers are included in this volume. The prize for the Best Paper was awarded to Yael Berkman and Ishay Haviv for their paper *Kernelization for H-Coloring*.

In addition, this volume contains the description of the PACE 2025 challenge as well as of the best performing solvers in the challenge. The PACE (Parameterized Algorithms and Computational Experiments) challenge has been hosted by IPEC since 2016 to emphasize the link between theory and practice. This year's problems were the classic Dominating Set and Hitting Set problems. Details of the challenge are given in an included article by the PACE 2025 program committee, consisting of Mario Grobler and Sebastian Siebertz (chair). The conference also hosted a PACE 2025 award session with brief presentations of the winning solvers and a PACE 2025 poster session.

The IPEC 2025 conference had two keynote talks. The first was an invited tutorial lecture by Martin Koutecký, titled *Parameterized Algorithms Via and For Block-Structured Integer Programs*. An invited article by Martin Koutecký, *A Brief History of Parameterized Algorithms for Block-Structured Integer Programs*, is included in this volume. The second keynote was by Jaroslav Nešetřil and Patrice Ossona de Mendez on *Sparsity and More*, as winners of the 2025 EATCS-IPEC Nerode Prize. The prize was awarded to them for four papers that laid the foundations of sparsity: an area of structural graph theory centered around the concepts of nowhere dense graph classes and graph classes with bounded expansion.

Finally, we wish to thank the program committee for the thorough reviews and discussions and to thank the external reviewers for their in-depth reviews. We also thank the authors who submitted their work, all participants for their interest, and the session chairs for guiding each session. We are grateful to the local organizers of ALGO 2025 for making the conference run as smoothly as it did and particularly to Paweł Rzążewski for his helpfulness and the fast communication.

Akanksha Agrawal and Erik Jan van Leeuwen
Madras and Utrecht, October 2025

20th International Symposium on Parameterized and Exact Computation (IPEC 2025).
Editors: Akanksha Agrawal and Erik Jan van Leeuwen
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The Tradition of I(W)PEC

2004	1st IWPEC,	Bergen, Norway	chairs: Rodney Downey, Michael Fellows, Frank Dehne
2006	2nd IWPEC,	Zürich, Switzerland	chairs: Hans L. Bodlaender, Michael A. Langston
2008	3rd IWPEC,	Victoria, Canada	chairs: Martin Grohe, Rolf Niedermeier
2009	4th IWPEC,	Copenhagen, Denmark	chairs: Jianer Chen, Fedor V. Fomin
2010	5th IPEC,	Chennai, India	chairs: Venkatesh Raman, Saket Saurabh
2011	6th IPEC,	Saarbrücken, Germany	chairs: Dániel Marx, Peter Rossmanith
2012	7th IPEC,	Ljubljana, Slovenia	chairs: Dimitrios Thilikos, Gerhard Woeginger
2013	8h IPEC,	Sophia Antipolis, France	chairs: Gregory Gutin, Stefan Szeider
2014	9th IPEC,	Wrocław, Poland	chairs: Marek Cygan, Pinar Heggernes
2015	10th IPEC,	Patras, Greece	chairs: Thore Husfeldt, Iyad Kanj
2016	11th IPEC,	Arrhus, Denmark	chairs: Jiong Guo, Danny Hermelin
2017	12th IPEC,	Vienna, Austria	chairs: Daniel Lokshtanov, Naomi Nishimura
2018	13th IPEC,	Helsinki, Finland	chairs: Christophe Paul, Michał Pilipczuk
2019	14th IPEC,	Munich, Germany	chairs: Bart Jansen, Jan Arne Telle
2020	15th IPEC,	~~Hong Kong~~ virtual event	chairs: Yixin Cao, Marcin Pilipczuk
2021	16th IPEC,	~~Lisbon, Portugal~~ virtual event	chairs: Petr Golovach, Meirav Zehavi
2022	17th IPEC,	Potsdam, Germany	chairs: Holger Dell, Jesper Nederlof
2023	18th IPEC,	Amsterdam, the Netherlands	chairs: Neeldhara Misra, Magnus Wahlström
2024	19th IPEC,	Egham, United Kingdom	chairs: Édouard Bonnet and Paweł Rzążewski
2025	20th IPEC,	Warsaw, Poland	chairs: Akanksha Agrawal and Erik Jan van Leeuwen

20th International Symposium on Parameterized and Exact Computation (IPEC 2025).
Editors: Akanksha Agrawal and Erik Jan van Leeuwen
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Committees

IPEC 2025 Program Committee

Akanksha Agrawal (IIT Madras, India) *(co-chair)*
Matthias Bentert (University of Bergen, Norway)
Fedor V. Fomin (University of Bergen, Norway)
Markus Hecher (Computer science Research Institute of Lens, France)
Danny Hermelin (Ben-Gurion University of the Negev, Israel)
Tanmay Inamdar (IIT Jodhpur, India)
Yusuke Kobayashi (Kyoto University, Japan)
Stefan Kratsch (Humboldt-Universität zu Berlin, Germany)
Alexandra Lassota (Eindhoven University of Technology, The Netherlands)
Daniel Lokshtanov (University of California Santa Barbara, USA)
Kitty Meeks (University of Glasgow, UK)
Fabrizio Montecchiani (University of Perugia, Italy)
Eunjin Oh (Pohang University of Science and Technology, South Korea)
Yota Otachi (Nagoya University, Japan)
Michał Pilipczuk (University of Warsaw, Poland)
Manuel Sorge (TU Wien, Austria)
Uéverton Souza (Universidade Federal Fluminense, Brazil)
Erik Jan van Leeuwen (Utrecht University, The Netherlands) *(co-chair)*
Jie Xue (New York University Shanghai, China)
Anna Zych-Pawlewicz (University of Warsaw, Poland)

PACE 2025 Program Committee

Mario Grobler (Universität Bremen, Germany)
Sebastian Siebertz (Universität Bremen, Germany) *(chair)*

2025 EATCS-IPEC Nerode Prize Committee

Russell Impagliazzo (University of California San Diego, USA)
Sang-il Oum (IBS, Korea) *(chair)*
Michał Pilipczuk (University of Warsaw, Poland)

2024-2025 IPEC Steering Committee

Akanksha Agrawal (IIT Madras, India) (2024–2027)
Hans L. Bodlaender (Utrecht University, The Netherlands) (2023–2026)
Édouard Bonnet (ENS Lyon, France) (2023–2026)
Łukasz Kowalik (University of Warsaw, Poland) (2022–2025)
Erik Jan van Leeuwen (Utrecht University, The Netherlands) (2024–2027)
Neeldhara Misra (IIT Gandhinagar, India) (2022–2025)
Paweł Rzążewski (University of Warsaw & Warsaw Univ. of Technology, Poland) (2023–2026)
Dimitrios M. Thilikos (LIRMM, France) (2024–2027)
Magnus Wahlström (Royal Holloway University of London, UK) (2022–2025) *(chair)*

20th International Symposium on Parameterized and Exact Computation (IPEC 2025).
Editors: Akanksha Agrawal and Erik Jan van Leeuwen
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

2024-2025 PACE Steering Committee

Max Bannach (European Space Agency, The Netherlands) (since 2023)
Sebastian Berndt (Universität zu Lübeck, Germany) (since 2023)
Holger Dell (IT University of Copenhagen, Denmark) (since 2016)
Bart M.P. Jansen (Eindhoven University of Technology, The Netherlands) (since 2016) *(chair)*
Philipp Kindermann (Universität Trier, Germany) (since 2024)
André Nichterlein (TU Berlin, Germany) (since 2021)
Christian Schulz (Universität Heidelberg, Germany) (since 2022)
Soeren Terziadis (Eindhoven University of Technology, The Netherlands) (since 2024)

List of External Reviewers

Oliver Bachtler
Tian Bai
Julia Baligacs
Max Bannach
Benjamin Bergougnoux
Sebastian Berndt
Davide Bilo
Joakim Blikstad
Marin Bougeret
Cornelius Brand
Tom-Lukas Breitkopf
Florent Capelli
Tianzhi Chen
Florian Chudigiewitsch
Alessio Conte
Alex Crane
Eva Deltl
Yann Disser
Riccardo Dondi
Eduard Eiben
Johannes K. Fichte
Alexander Firbas
Till Fluschnik
Jaroslav Garvardt
Tatsuya Gima
Andreas Göbel
Petr Golovach
Timothy Gomez
Carla Groenland
Roland Guttenberg
Thekla Hamm
Tesshu Hanaka
Meike Hatzel
Anton Herrmann
Yuval Itzhaki
Satyabrata Jana
Lawqueen Kanesh
Matthias Kaul
Leon Kellerhals
Evangelos Kipouridis
Dušan Knop
Tomohiro Koana
Yasuaki Kobayashi
Sudeshna Kolay
Balagopal Komarath
Christian Komusiewicz
Kenneth Langedal
Laura Larios-Jones
Jayakrishnan Madathil
Yasir Mahmood
Kevin Mann
Eli Mitchell
Hendrik Molter
Yash More
Wojciech Nadara
Alfredo Navarra
André Nichterlein
Yuto Okada
Karolina Okrasa
Fahad Panolan
Daniel Perz
Théo Pierron
Marcin Pilipczuk
Lars Rohwedder
Ignaz Rutter
Abhishek Sahu
Dvir Shabtay
Roohani Sharma
Sofia Simola
Frank Sommer
Ramanujan M.Sridharan
Joakim Sunde
Suguru Tamaki
Yuma Tamura
Alessandra Tappini
Karol Węgrzycki
Simon Wietheger
Michał Włodarczyk
Jatin Yadav
Meirav Zehavi
Chihao Zhang

List of Authors

Aditya Anand (26)
University of Michigan, Ann Arbor, MI, USA

Leonid Antipov (6)
Humboldt-Universität zu Berlin, Germany

Max Bannach (39)
European Space Agency, AI and Data Science Section, Noordwijk, The Netherlands

Jesse Beisegel (30)
Institute of Mathematics, Brandenburg University of Technology, Cottbus, Germany

Benjamin Bergougnoux (31)
Aix-Marseille Université, CNRS, LIS, Marseille, France

Yael Berkman (5)
The Academic College of Tel Aviv-Yaffo, Tel Aviv, Israel

Václav Blažej (18)
University of Warsaw, Poland

Hans L. Bodlaender (20)
Utrecht University, The Netherlands

Narek Bojikian (19)
Humboldt-Universität zu Berlin, Germany

Édouard Bonnet (17)
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Marin Bougeret (23)
LIRMM, Université de Montpellier, Montpellier, France

Christina Büsing (7)
Research and Teaching Area for Combinatorial Optimization, RWTH Aachen University, Germany

Guilherme C. M. Gomes (23)
LIRMM, Université de Montpellier, CNRS, Montpellier, France; Universidade Federal de Minas Gerais, Belo Horizonte, Brazil

Florian Chudigiewitsch (39)
Institute for Theoretical Computer Science, University of Lübeck, Germany

Vincent Cohen-Addad (26)
Google Research, NYC, USA

Tommaso D'Orsi (26)
Bocconi University, Milan, Italy

Guilherme D. da Fonseca (34)
LIS, Aix-Marseille Université, France

Jona Dirks (33)
Université Clermont-Auvergne, CNRS, Mines de Saint-Étienne, Clermont-Auvergne-INP, LIMOS, 63000 Clermont-Ferrand, France

Alexander Dobler (35)
Algorithms and Complexity Group, TU Wien, Austria

Vinicius F. dos Santos (23)
Universidade Federal de Minas Gerais, Belo Horizonte, Brazil

Maurice Draeger (7)
RWTH Aachen University, Germany

Joanne Dumont (13)
Université d'Orléans, INSA CVL, LIFO, UR 4022, Orléans, France

Michelle Döring (10)
Hasso Plattner Institute, University of Potsdam, Germany

Jan Fehse (10)
Hasso Plattner Institute, University of Potsdam, Germany

Fabien Feschet (34)
LIMOS, Université Clermont Auvergne, France

Simon Dominik Fink (35)
Algorithms and Complexity Group, TU Wien, Austria

Florian Fontan (36)
Independent researcher, Grenoble, France

Tobias Friedrich (8, 10)
Hasso Plattner Institute (HPI), University of Potsdam, Germany

Marcelo Garlet Milani (2)
National Institute of Informatics, Tokyo, Japan

Yan Gerard (34)
LIMOS, Université Clermont Auvergne, France

Enna Gerhard (33)
University of Bremen, Bibliothekstraße 5, 28359 Bremen, Germany

Tatsuya Gima (29)
Hokkaido University, Sapporo, Japan

20th International Symposium on Parameterized and Exact Computation (IPEC 2025).
Editors: Akanksha Agrawal and Erik Jan van Leeuwen
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Alexander Grigoriev (21)
Department of Data Analytics and Digitalisation, Maastricht University, The Netherlands

Mario Grobler (32)
University of Bremen, Germany

Anupam Gupta (26)
New York University, NY, USA

Sushmita Gupta (14)
The Institute of Mathematical Sciences, HBNI, Chennai, India

Tesshu Hanaka (4)
Kyushu University, Fukuoka, Japan

Ishay Haviv (5)
The Academic College of Tel Aviv-Yaffo, Tel Aviv, Israel

Steef Hegeman (27)
Leiden University, The Netherlands

Anton Herrmann (12)
Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

Takashi Horiyama (25)
Faculty of Information Science and Technology, Hokkaido University, Sapporo, Japan

Lars Jaffke (31)
NHH Norwegian School of Economics, Bergen, Norway

Pallavi Jain (14)
IIT Jodhpur, India

Satyabrata Jana (18)
University of Warwick, Coventry, UK

Bart M. P. Jansen (9)
Eindhoven University of Technology, The Netherlands

Klaus Jansen (15)
Department of Computer Science, Kiel University, Germany

Mohamed Jiddou (22)
LI-PARAD, Université de Versailles Saint-Quentin-En-Yvelines, Paris-Saclay, France

Mark Jones (11)
TU Delft, The Netherlands

Kai Kahler (15)
Department of Computer Science, Kiel University, Germany

Victoria Kaial (33)
Université Clermont-Auvergne, CNRS, Mines de Saint-Étienne, Clermont-Auvergne-INP, LIMOS, 63000 Clermont-Ferrand, France

Ken-ichi Kawarabayashi (2)
National Institute of Informatics, Tokyo, Japan

Katharina Klost (30)
Institute of Computer Science, Freie Universität Berlin, Germany

Kristin Knorr (30)
Institute of Computer Science, Freie Universität Berlin, Germany

Yasuaki Kobayashi (21, 29)
Hokkaido University, Sapporo, Japan

Christian Komusiewicz (12)
Institute of Computer Science, Friedrich Schiller University Jena, Germany

Martin Koutecký (1)
Computer Science Institute, Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic

Stefan Kratsch (6, 19)
Humboldt-Universität zu Berlin, Germany

Alfons Laarman (27)
Leiden University, The Netherlands

Jeroen S. K. Lamme (9)
Eindhoven University of Technology, The Netherlands

Michael Lampis (3, 13)
Université Paris-Dauphine, PSL University, CNRS UMR7243, LAMSADE, Paris, France

Euiwoong Lee (26)
University of Michigan, Ann Arbor, MI, USA

Mathieu Liedloff (13)
Université d'Orléans, INSA CVL, LIFO, UR 4022, Orléans, France

Nicola Lorenz (2)
Universität Hamburg, Germany

Lucas Lorieau (33)
Université Clermont-Auvergne, CNRS, Mines de Saint-Étienne, Clermont-Auvergne-INP, LIMOS, 63000 Clermont-Ferrand, France

Canhui Luo (40)
Huazhong University of Science and Technology, Wuhan, China

Zhipeng Lü (40)
Huazhong University of Science and Technology, Wuhan, China

George Manoussakis (22)
LI-PARAD, Université de Versailles Saint-Quentin-En-Yvelines, Paris-Saclay, France

Paula Marten (10)
Hasso Plattner Institute, University of Potsdam, Germany

Jan Martens (27)
Leiden University, The Netherlands

Corinna Mathwieser (7)
Research and Teaching Area for Combinatorial Optimization, RWTH Aachen University, Germany

Niklas Mohrin (10)
Hasso Plattner Institute, University of Potsdam, Germany

Nils Morawietz (12)
Institute of Computer Science, Friedrich Schiller University Jena, Germany; LaBRI, Université de Bordeaux, Talence, France

Daniel Neuen (17)
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Yuto Okada (29)
Nagoya University, Japan

Yuto Okura (25)
Graduate School of Information Science and Technology, Hokkaido University, Sapporo, Japan

Yota Otachi (29)
Nagoya University, Japan

Debmalya Panigrahi (26)
Duke University, Durham, NC, USA

Sijin Peng (26)
Massachusetts Institute of Technology, Cambridge, MA, USA

Anthony Perez (13)
Université d'Orléans, INSA CVL, LIFO, UR 4022, Orléans, France

Michał Pilipczuk (24)
Institute of Informatics, University of Warsaw, Poland

Lis Pirotton (15)
Department of Computer Science, Kiel University, Germany

M. S. Ramanujan (18)
University of Warwick, Coventry, UK

Fabienne Ratajczak (30)
Institute of Mathematics, Brandenburg University of Technology, Cottbus, Germany

Mathis Rocton (35)
Algorithms and Complexity Group, TU Wien, Austria

Ignasi Sau (23)
LIRMM, Université de Montpellier, CNRS, Montpellier, France

Robert Scheffler (30)
Institute of Mathematics, Brandenburg University of Technology, Cottbus, Germany

Jannik Schestag (11)
TU Delft, The Netherlands

André Schidler (37)
University of Freiburg, Germany

Sylvain Schmitz (24)
Université Paris Cité, CNRS, IRIF, France

Sanjay Seetharaman (14)
The Institute of Mathematical Sciences, HBNI, Chennai, India

Kazuhisa Seto (25)
Faculty of Information Science and Technology, Hokkaido University, Sapporo, Japan

Sebastian Siebertz (32)
University of Bremen, Germany

Kirill Simonov (8, 10)
Department of Informatics, University of Bergen, Norway

Henry Sinclair-Banks (24)
Institute of Informatics, University of Warsaw, Poland

Farehe Soheil (8, 10)
Hasso Plattner Institute (HPI), University of Potsdam, Germany

Marek Sokołowski (17)
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Frank Sommer (12)
Institute of Logic and Computation, TU Wien, Austria; Institute of Computer Science, Friedrich Schiller University Jena, Germany

Jacob Stegemann (2)
Universität Hamburg, Germany

Peter Strulo (18)
University of Warwick, Coventry, UK

Zhouxing Su (40)
Huazhong University of Science and Technology, Wuhan, China

Sylwester Swat (38)
Institute of Computing Science, Poznan University of Technology, Poland

Hayato Takaike (29)
Nagoya University, Japan

Prafullkumar Tale (28)
Indian Institute of Science Education and Research Pune, India

Hisao Tamaki (21)
Meiji University, Kawasaki, Japan

Junichi Teruyama (25)
Graduate School of Information Science, University of Hyogo, Kobe, Japan

Jakob Timm (10)
Hasso Plattner Institute, University of Potsdam, Germany

Ioan Todinca (13)
Université d'Orléans, INSA CVL, LIFO, UR 4022, Orléans, France

Daisuke Tsuru (4)
Kyushu University, Fukuoka, Japan

Malte Tutas (15)
Department of Computer Science, Kiel University, Germany

Tom C. van der Zanden (21)
Department of Data Analytics and Digitalisation, Maastricht University, The Netherlands

Manolis Vasilakis (3)
Université Paris-Dauphine, PSL University, CNRS UMR7243, LAMSADE, Paris, France

Guillaume Verger (36)
Independent researcher, Lyon, France

Ruben F. A. Verhaegh (9)
Eindhoven University of Technology, The Netherlands

Shaily Verma (10)
Hasso Plattner Institute, University of Potsdam, Germany

Jonne Visser (20)
Utrecht University, The Netherlands

Marcel Wienöbst (39)
Institute for Theoretical Computer Science, University of Lübeck, Germany

Michał Włodarczyk (16)
University of Warsaw, Poland

Meirav Zehavi (14)
Ben-Gurion University of the Negev, Beer-Sheva, Israel

Qingyun Zhang (40)
Huazhong University of Science and Technology, Wuhan, China

A Brief History of Parameterized Algorithms for Block-Structured Integer Programs

Martin Koutecký
Computer Science Institute, Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic

Abstract

Integer Programming (IP) is a fundamental but computationally hard problem. Still, certain efficiently solvable subclasses have been identified over time, most notably totally unimodular IPs in the 1950s, and fixed-dimension IPs in the 1980s. Starting around the year 2000, a stream of research has identified block-structured IPs as yet another tractable subclass. In this paper, we give a brief and incomplete review of this history, with a focus on several of the author's contributions.

2012 ACM Subject Classification Theory of computation → Convex optimization

Keywords and phrases Integer Programming, Parameterized Algorithm, Graver Basis, Treedepth, n-fold, tree-fold, 2-stage stochastic, multistage stochastic, Mixed-Integer Programming

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.1

Category Invited Talk

Related Version This paper is partially based on my habilitation. It contains some material which was omitted here due to space limitations, in particular an entire chapter on the papers [11, 13].
Habilitation: `https://research.koutecky.name/papers/habil.pdf`

Funding Partially supported by Charles Univ. project UNCE 24/SCI/008, by the ERC-CZ project LL2406 of the Ministry of Education of Czech Republic, and by the project 24-10306S of GA ČR.

1 Introduction

The **integer programming** problem is

$$\min\{f(\mathbf{x}) \mid A\mathbf{x} = \mathbf{b}, \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}, \mathbf{x} \in \mathbb{Z}^n\}, \tag{IP}$$

with A an integer $m \times n$ matrix, $f : \mathbb{R}^n \to \mathbb{R}$ a separable convex function, $\mathbf{b} \in \mathbb{Z}^m$, and $\mathbf{l}, \mathbf{u} \in (\mathbb{Z} \cup \{\pm\infty\})^n$. IP is well-known to be strongly NP-hard (shown by Karp [64]) already in the special case when $f(\mathbf{x}) = \mathbf{w}\mathbf{x}$ is a linear objective function for some vector $\mathbf{w} \in \mathbb{Z}^n$:

$$\min\{\mathbf{w}\mathbf{x} \mid A\mathbf{x} = \mathbf{b}, \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}, \mathbf{x} \in \mathbb{Z}^n\}. \tag{ILP}$$

Despite this, integer programming can be often solved efficiently in practice and has become a central optimization paradigm in many application domains, e.g., problems related to planning [97], vehicle routing [95], process scheduling [39], packing [76], and network hub location [1]. To get a feeling for the breadth of the applicability of IP, one can for example visit the Case Studies[1] page of the Gurobi solver.

Because of its importance, IP has been studied from many perspectives, including heuristics [47], or methods such as cutting planes [78], branch-and-cut [99, Section 9.6], etc. We are interested in identifying broad subclasses of IP which can be solved to exact optimality with provable time complexity guarantees. To do so, we take the perspective of *parameterized*

[1] `https://www.gurobi.com/case_studies/`

© Martin Koutecký;
licensed under Creative Commons License CC-BY 4.0

complexity [22]: given an instance with encoding length L, we want to understand how the running time of an algorithm scales with the size of the input L and with some additional parameter k of the instance. An algorithm is said to be *fixed-parameter tractable* (FPT) if its running time is bounded by $g(k)\text{poly}(L)$ for some computable function g. A problem which is W[1]-*hard* is unlikely to admit an FPT algorithm, and in this case, one seeks the weaker notion of a *slice-wise polynomial* (XP) algorithm, whose complexity is bounded by $L^{g(k)}$. Note that the FPT vs. XP difference provides a finer distinction between algorithms which are both "polynomial for fixed k", a common claim in the literature, and indeed an important part of our work (not covered here) has been deciphering which algorithms in the literature are FPT, and which are (only) XP [44]. A problem is unlikely to admit an XP algorithm if it is para-NP-hard, meaning that it is NP-hard already for some constant value of the parameter. Another important notion is that of a *strongly polynomial algorithm*, which is an algorithm that makes a number of arithmetic operations independent of the magnitude of numbers on input; in the case of IP, this means that the number of arithmetic operations is bounded by a function of n only.

Before moving on to the tractable classes of IP studied by us, let us briefly mention three other important "islands of tractability" studied in the literature. Already in 1956, Hoffman and Kruskal [57] have shown that if A is totally unimodular, that is, all of its subdeterminants are -1, 0, or 1, then ILP coincides with its continuous relaxation[2]. Combined with the polynomiality of linear programming, this implies that ILP can be solved in polynomial time when A is totally unimodular. A natural generalization of totally unimodular matrices are Δ-*modular* matrices, whose subdeterminants are between integers $-\Delta$ and Δ for some fixed Δ. Artmann et al. [3] have shown that ILP is strongly polynomial when A is 2-modular, and other partial results are known [38,48,81], but the question of whether ILP with a Δ-modular matrix is FPT parameterized by Δ remains open.

A different tractable class of IP is formed by instances with a small number n of variables, as first shown by Lenstra [75]. His algorithm was subsequently improved by Kannan [62] and generalized by Grötschel, Lovász, and Schrijver [49] to optimize any convex function over the integers contained in any convex set. The state-of-the-art is due to Reis and Rothvoss [90] who have shown that ILP can be solved in time $(\log n)^{\mathcal{O}(n)}\text{poly}(L)$ (as before, L is the encoding length of the input instance); the important question whether an algorithm with complexity $2^{\mathcal{O}(n)}\text{poly}(L)$ exists remains open.

About the same time as Lenstra's result, Papadimitriou [86] has shown that the textbook dynamic programming algorithm for Knapsack can be generalized to ILP, achieving complexity $n^{\mathcal{O}(m)} \cdot (m\|A, \mathbf{b}\|_\infty)^{\mathcal{O}(m^2)}$, where $\|A, \mathbf{b}\|_\infty$ is the maximum absolute value of any entry appearing in A and $\mathbf{b}$. Looking back, it is not too difficult to tweak this algorithm to run in time $n \cdot (m\|A\|_\infty)^{\mathcal{O}(m^2)}\text{poly}(L)$, which is FPT parameterized by the combined parameter $\|A\|_\infty + m$, that is, when A has few rows and small coefficients in absolute value. However, this complexity has only been stated explicitly recently [71].

1.1 Block-structured Integer Programs

Our focus is on how a stream of research spanning over two decade has blossomed into a unified theory including significant time complexity improvements, generalizations to wider classes, and complementary lower bound results. Besides the results discussed in this paper, we have also shown how this theory can be used to obtain efficient algorithms for problems in scheduling, stringology, graph theory, and computational social choice [36, 44, 67–72, 74].

[2] The continuous relaxation of an ILP instance is the same instance, just without the $\mathbf{x} \in \mathbb{Z}^n$ constraint. This is then an instance of *linear programming*.

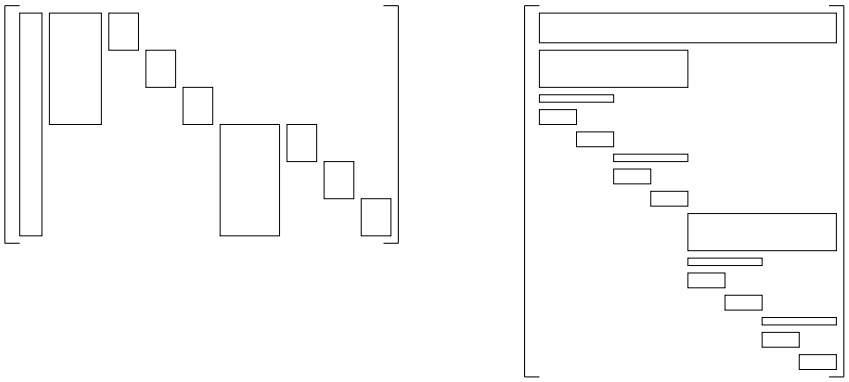


Figure 1 On the left, a schematic multi-stage with three levels is presented. On the right, a schematic tree-fold with 4 layers is pictured. All entries within a rectangle can be non-zero, all entries outside of the rectangles must be zero.

The purpose of this section is to introduce two important classes of block-structured IPs, and briefly explain the motivation and history of their study. A matrix $A^{(n)}$ is *n-fold* with blocks A_1, A_2 if it has the form

$$A^{(n)} = \begin{pmatrix} A_1 & A_1 & \cdots & A_1 \\ A_2 & & & \\ & A_2 & & \\ & & \ddots & \\ & & & A_2 \end{pmatrix}, \tag{1}$$

for $A_1 \in \mathbb{Z}^{r \times t}$ and $A_2 \in \mathbb{Z}^{s \times t}$. To be precise, the above may be called a "uniform" n-fold matrix, as opposed to a "generalized" n-fold matrix in which the blocks may differ. We choose to deal with the uniform case here because:

(a) it simplifies exposition,
(b) the key proof ideas usually extend to the generalized case,
(c) the uniform case is usually without loss of generality, because the generalized case can be embedded in it by suitable encoding tricks (though incurring some inefficiencies).

On the other hand, the general case is more appropriate when dealing with applications.

A *2-stage* matrix $B^{(n)}$ is the transpose of an n-fold matrix, i.e.,

$$B^{(n)} = \begin{pmatrix} A_1 & A_2 & & \\ \vdots & & \ddots & \\ A_1 & & & A_2 \end{pmatrix}, \tag{2}$$

We will also consider generalizations of both classes, so called *tree-fold* and *multi-stage stochastic* IPs, whose matrices have a recursive structure. For example, a tree-fold matrix has the form (1), except each block A_2 is itself a tree-fold matrix; see Figure 1.

1.2 N-fold Integer Programming

N-fold IP was introduced under this name by De Loera et al. [26] in 2008, but programs of this form have appeared in the literature much earlier. Consider the transportation problem, which asks for an optimal routing from several sources to several destinations. It has been defined by Hitchcock [54] in 1941 and independently studied by Kantorovich [63] in 1942,

and Dantzig [23] studied it in 1951 in the context of the applicability of the simplex method. The transportation problem may be seen as a *table problem* where we are given m row-sums and n column-sums and the task is to fill in non-negative integers into the table so as to satisfy the given row- and column-sums. A natural generalization to higher-dimensional tables, called *multiway tables*, has been studied already in 1947 by Motzkin [80]. It also has applications in privacy in databases and confidential data disclosure of statistical tables, see a survey by Fienberg and Rinaldo [37] and the references therein.

Specifically, the three-way table problem is to decide if there exists a non-negative integer $l \times m \times n$ table satisfying given line-sums, and to find the table if there is one. Deciding the existence of such a table is NP-complete already for $l = 3$ [27]. Moreover, every bounded integer program can be isomorphically represented in polynomial time for some m and n as some $3 \times m \times n$ table problem [28]. The complexity with l, m parameters and n variable thus became an interesting problem. Let the input line-sums be given by vectors $\mathbf{u} \in \mathbb{Z}^{ml}, \mathbf{v} \in \mathbb{Z}^{nl}$ and $\mathbf{w} \in \mathbb{Z}^{nm}$. Observe that the problem can be formulated as an IP with variables $x^i_{j,k}$ for $i \in [n]$, $j \in [m]$ and $k \in [l]$, objective function $f \equiv 0$, and the following constraints:

$$\begin{aligned}
\sum_{i=1}^{n} x^i_{j,k} &= u_{j,k} && \forall j \in [m], k \in [l], \\
\sum_{j=1}^{m} x^i_{j,k} &= v^i_k && \forall i \in [n], k \in [l], \\
\sum_{k=1}^{l} x^i_{j,k} &= w^i_j && \forall i \in [n], j \in [m], \\
\mathbf{x} &\geq \mathbf{0} && .
\end{aligned}$$

Let I_k be the $k \times k$, $k \in \mathbb{N}$, identity matrix, and $\mathbf{1}_k$ the all-ones vector of dimension $k \in \mathbb{N}$. Then the above, written in matrix form, becomes $A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0}$ with $\mathbf{b} = (\mathbf{u}, \mathbf{v}^1, \mathbf{w}^1, \mathbf{v}^2, \mathbf{w}^2, \dots, \mathbf{v}^n, \mathbf{w}^n)$, and with

$$A = \begin{pmatrix} I_{ml} & I_{ml} & \cdots & I_{ml} \\ J & & & \\ & J & & \\ & & \ddots & \\ & & & J \end{pmatrix}, \text{ where } J = \begin{pmatrix} I_l & \cdots & I_l \\ \mathbf{1}_l & & \\ & \ddots & \\ & & \mathbf{1}_l \end{pmatrix} \in \mathbb{Z}^{(l+m)\times ml} .$$

Here, J has m diagonal blocks $\mathbf{1}_l$ and A has n diagonal blocks J. Thus, A can be viewed either as an n-fold matrix, or as a tree-fold matrix with 3 levels.

A different implicit appearance of the n-fold structure is in the famous[3] 1961 and 1963 papers by Gilmore and Gomory [45,46] on solving the Cutting Stock problem. Their work has been essential in regards to fundamental notions like column generation, cutting planes, and the *Configuration LP* whose depiction from the 1963 paper [46] appears in Figure 2. It is clear that the presented matrix is n-fold, and we will return to it in Section 3. In fact, the significance of the n-fold structure in packing and scheduling problems seems to have been lost until a brief mention in the paper introducing n-fold IP [26], and has only been taken up in earnest [14, 15, 50, 60, 65, 68, 69] in the wake of the paper of Knop and Koutecký from 2018 [67]. Let us now briefly describe this connection.

[3] Over 3,000 and 1,800 citations, respectively, according to Google Scholar, as of October 2025.

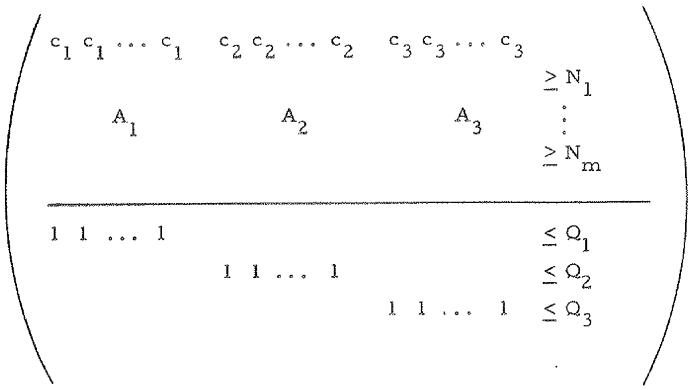


Figure 2 A figure from "A Linear Programming Approach to the Cutting-Stock Problem – Part II" by Gilmore and Gomory [46] depicting the Configuration LP (not yet under this name) of the "Machine Balance Problem".

The problem of *uniformly related machines makespan minimization*, denoted $Q||C_{\max}$ in the standard notation, is the following. We are given m machines, each with speed $s_i > 0$, and n jobs, where the j-th job has processing time $p_j \in \mathbb{N}$ and processing it on machine i takes time p_j/s_i. The task is to assign jobs to machines such that the time when the last job finishes (the *makespan*) is minimal, i.e., if M_i is the set of jobs assigned to machine i and $\bigcup_i M_i$ contains all jobs, the task is to minimize $\max_{i\in[m]} \sum_{j\in M_i} p_j/s_i$. The decision version of the problem asks whether there is a schedule of makespan $C_{\max} \in \mathbb{R}$. We consider the scenario when $p_{\max} = \max_j p_j$ is bounded by a parameter and the input is represented *succinctly* by multiplicities $n_1, \ldots, n_{p_{\max}}$ of jobs of each length, i.e., n_ℓ is the number of jobs with $p_j = \ell$. Letting x^i_j be a variable representing the number of jobs of length j assigned to machine i, Knop and Koutecký [67] give the following n-fold formulation:

$$\sum_{i=1}^{m} x^i_j = n_j \qquad \forall j \in [p_{\max}], \tag{3}$$

$$\sum_{j=1}^{p_{\max}} j \cdot x^i_j \leq \lfloor s_i \cdot C_{\max} \rfloor \qquad \forall i \in [m] \ . \tag{4}$$

Constraints (3) ensure that each job is scheduled on some machine, and constraints (4) ensure that each machine finishes before time $C_{\max}$. This corresponds to an n-fold formulation with $A_1 = I_{p_{\max}}$ and $A_2 = (1, 2, \ldots, p_{\max})$ and with $\|A^{(n)}\|_\infty = p_{\max}$.

Another scheduling problem is finding a schedule minimizing the *sum of weighted completion times* $\sum w_j C_j$. Knop and Koutecký [67] show an n-fold formulation for this problem as well, in particular one which has a separable quadratic objective. In the context of scheduling, what sets methods based on n-fold IP apart is that they allow the handling of many "types" of machines (such as where machines have different speeds) and also "non-linear" objectives (such as the quadratic objective in the formulation for $\sum w_j C_j$).

Another field where n-fold IP has had an impact is computational social choice. The problem of Bribery asks for a cheapest manipulation of voters which lets a particular candidate win an election. An FPT algorithm was known for Bribery parameterized by the number of candidates which relied on Lenstra's algorithm. However, this approach has two downsides: a time complexity which is doubly-exponential in the parameter, and the fact

that voters have to be "uniform" and cannot have distinct cost functions. Knop et al. [72] resolved this problem using n-fold IP by showing a single-exponential algorithm for many Bribery-type problems, even in the case when each voter has a different cost function.

Lastly, the Dantzig-Wolfe decomposition [24] is an fundamental algorithm for solving linear programs with an n-fold structure which decomposes the problem into a master problem and a series of subproblems corresponding to the individual blocks. While this approach is primarily studied in the context of continuous optimization, it can be fruitfully generalized to integer programming as well [98].

1.3 2-stage Stochastic Integer Programming

The motivation for the study of 2-stage stochastic matrices comes from decision making under uncertainty. Here, one is asked to make a partial decision in a "first stage", and after realization of some random data, one has to complete their decision in a "second stage". The goal is minimizing the "direct" cost of the first-stage decision plus the expected cost of the second-stage decision. Random data are often modeled by a finite set of n *scenarios*, each with a given probability. Assume that the scenarios are represented by integer vectors $\mathbf{b}^1, \dots, \mathbf{b}^n \in \mathbb{Z}^t$, their probabilities by $p_1, \dots, p_n \in (0, 1]$, the first-stage decision is encoded by a variable vector $\mathbf{x}^0 \in \mathbb{Z}^r$, and the second-stage decision for scenario $j \in [n]$ is encoded by a variable vector $\mathbf{x}^j \in \mathbb{Z}^s$. Setting $\mathbf{x} := (\mathbf{x}^0, \mathbf{x}^1, \dots, \mathbf{x}^n)$ and $\mathbf{b} := (\mathbf{b}^1, \dots, \mathbf{b}^n)$ then makes it possible to write this problem as

$$\min \mathbf{w}^0\mathbf{x}^0 + \sum_{j=1}^{n} p_j \mathbf{w}'\mathbf{x}^j : B^{(n)}\mathbf{x} = \mathbf{b},\ \mathbf{l} \le \mathbf{x} \le \mathbf{u},\ \mathbf{x} \in \mathbb{Z}^{r+ns}, \text{ where } B^{(n)} = \begin{pmatrix} A_1 & A_2 & & \\ \vdots & & \ddots & \\ A_1 & & & A_2 \end{pmatrix}, \tag{5}$$

with $A_1 \in \mathbb{Z}^{t \times r}$, $A_2 \in \mathbb{Z}^{t \times s}$, and $\mathbf{l}, \mathbf{u} \in \mathbb{Z}^{r+ns}$ some lower and upper bounds. Problem (5) is called 2-*stage stochastic IP* and finds many applications [7, 61, 88].

Similarly to the Dantzig-Wolfe decomposition method for problems with an n-fold structure, a decompositional method for problems with a 2-stage structure has been introduced by Benders [5] in 1962. Where Dantzig-Wolfe introduces new columns in each iteration (leading to "column generation"), Benders introduces new rows (leading to "row generation"). A generalization to the (mixed) integer case has been also studied [94].

Integer programs with block structure have also been the subject of at least two habilitations, that of Martin [79] and Nowak [83], which contain many other relevant references.

2 (Strongly) Fixed-parameter Tractable Algorithms for ILP and IP

This section is based on the papers [73] and [31]. Note that even though [73] has been published in 2018 and [31] only in 2024, the results of both papers have been obtained in 2017-2018 and both logically as well as chronologically precede the papers reviewed in subsequent sections.

2.1 State of the Art in 2018

We will now review the developments regarding parameterized complexity of block-structured IPs until 2018. Whenever we talk about FPT algorithms here, the parameter is the largest coefficient in absolute value $\|A\|_\infty$, and the sizes of the blocks composing the block-structured matrix at hand. The key notion behind most of the results reviewed here is the *Graver basis*

of the matrix A, $\mathcal{G}(A)$, which is the set of non-zero integer vectors $\mathbf{g}$ satisfying $A\mathbf{g} = \mathbf{0}$ that cannot be decomposed into two non-zero integer vectors with the same sign-pattern (i.e., belonging to the same orthant) satisfying the same equation. These *Graver basis elements* are thus called *indecomposable*, *irreducible*, or *conformal-minimal*. The Graver basis allows a simple *iterative augmentation* procedure: given a feasible solution $\mathbf{x}$, either $\mathbf{x} + \mathbf{g}$ for some $\mathbf{g} \in \mathcal{G}(A)$ is feasible and improves the objective function, or $\mathbf{x}$ is already optimal.

We now discuss three areas where the main developments took place.

2-stage and multi-stage stochastic IP. Already in 2003, Hemmecke and Schultz have shown that 2-stage stochastic IP is FPT [53], although this FPT bound was not explicit. At the heart of their algorithm is the following finiteness result about $\mathcal{G}(A)$. For a number n, 2-stage stochastic matrix $B^{(n)}$, and a vector $\mathbf{x} = (\mathbf{x}^0, \mathbf{x}^1, \ldots, \mathbf{x}^n)$, call the subvector $\mathbf{x}^0$ the *global brick*, and the subvectors $\mathbf{x}^1, \ldots, \mathbf{x}^n$ the *local bricks*. Then, there exists a number n_0 such that for every $n \geq n_0$, the set of possible global and local bricks that appear in any $\mathbf{g} \in \mathcal{G}(B^{(n)})$ is fixed and independent of n. Moreover, they provide an algorithm to compute this finite set of bricks. A simple branching algorithm then allows one to "guess" the global brick, compute, for each block, the optimal "matching" local brick, and in this way find an optimal Graver basis element. Combining this with the iterative augmentation procedure then yields an FPT algorithm. (This algorithm may take a long time to converge, but improving convergence is a separate and independent topic which we leave aside for now.) Since the aforementioned finiteness result is obtained by a saturation argument [77] and does not provide an explicit upper bound, there was no explicit upper bound on the complexity of the algorithm. This result, using an analogous technique, was extended to multi-stage stochastic IP by Aschenbrenner and Hemmecke [4].

***N*-fold and tree-fold IP.** The story was more complicated for n-fold IP. Call the subvectors $\mathbf{x}^1, \ldots, \mathbf{x}^n$ of an nt-dimensional vector $\mathbf{x}$ *bricks*. N-fold IP has been introduced by De Loera et al. [26] in 2008, and it is in this paper where the fundamental structural result about $\mathcal{G}(A^{(n)})$ has been stated: the number of non-zero bricks of any $\mathbf{g} \in \mathcal{G}(A^{(n)})$ is independent of n, and so are their values. The arguments needed for this claim have been already developed in 2003 (by Santos and Sturmfels [91]) and 2007 (by Hoşten and Sullivant [58]). A particularly ingenious trick appears in the paper by Santos and Sturmfels [91], where they show that the structure of the Graver basis of $\mathcal{G}(A^{(n)})$ can be understood through what is essentially *the Graver basis of the Graver basis of* A_2*:* $\mathcal{G}(A_1\mathcal{G}(A_2))$. Quoting from [91], *"The phrase 'the Graver basis of the Graver basis' is not a typo but it is the punchline".*

Still, the 2008 algorithm of [26] only had complexity $n^{g(A_1,A_2)}\text{poly}(L)$, where g is some computable function, so this is an XP, not FPT, algorithm. A breakthrough was achieved by Hemmecke, Onn, and Romanchuk [52] in 2013, where they combined the ideas from [26] with a Knapsack-like dynamic programming algorithm (in the spirit of Papadimitriou [86]) to obtain the first FPT algorithm for n-fold IP. This algorithm was highlighted in the textbook of Downey and Fellows [29], and it attracted much attention after we have shown its wide applicability in scheduling [67], computational social choice [72], and other areas [71]. Spurred by these results, Chen and Marx [14] have introduced tree-fold IP in early 2018, and have shown that it is FPT as well.

Graph Parameters. The notion of sparsity has enjoyed tremendous success in graph theory [82, 87], and it is natural to ask about its applicability to IP. We focus on two graphs associated with A:

- the *primal graph* $G_P(A)$, which has a vertex for each column and two vertices are connected if there exists a row such that both columns are non-zero, and,
- the *dual graph* $G_D(A) = G_P(A^\intercal)$, which is the above with rows and columns swapped.

Two standard parameters of structural sparsity are the *treewidth* (measuring the "tree-likeness" of a graph) and the more restrictive treedepth (measuring its "star-likeness"). We focus on the latter here: the *treedepth* of a graph G denoted $\mathrm{td}(G)$ is the smallest height of a rooted forest F such that each edge of G is between vertices which are in a descendant-ancestor relationship in F. The *primal treedepth of* A is $\mathrm{td}_P(A) := \mathrm{td}(G_P(A))$, and analogously the *dual treedepth of* A is $\mathrm{td}_D(A) := \mathrm{td}(G_D(A))$. We will not define treewidth, but we shall denote the treewidth of $G_P(A)$ and $G_D(A)$ by $\mathrm{tw}_P(A)$ and $\mathrm{tw}_D(A)$, respectively, and we note that bounded treedepth implies bounded treewidth but not vice versa.

It follows from Freuder's algorithm [41] and was reproven by Jansen and Kratsch [59] that IP is FPT parameterized by the primal treewidth $\mathrm{tw}_P(A)$ and the largest variable domain $\|\mathbf{u} - \mathbf{l}\|_\infty$. Regarding the dual graph $G_D(A)$, the parameters $\mathrm{td}_D(A)$ and $\mathrm{tw}_D(A)$ were considered by Ganian et al. [43]. They show that even deciding feasibility of IP is NP-hard on instances with $\mathrm{tw}_I(A) = 3$ ($\mathrm{tw}_I(A)$ denotes the treewidth of the *incidence graph*; $\mathrm{tw}_I(A) \leq \mathrm{tw}_D(A) + 1$ always holds). Furthermore, they show that IP is FPT parameterized by $\mathrm{tw}_I(A)$ and parameter Γ, which is an upper bound on any prefix sum of $A\mathbf{x}$ for any feasible solution $\mathbf{x}$. Dvořák et al. [30] introduce the parameter fracture number. Having a bounded *variable fracture number* $\mathfrak{p}^V(A)$ implies that deleting a few columns of A breaks it into independent blocks of small size, similarly for *constraint fracture number* $\mathfrak{p}^C(A)$ and deleting a few rows. Having a bounded $\mathfrak{p}^V(A)$ is essentially equivalent to having a generalized n-fold structure, and similarly having a bounded $\mathfrak{p}^C(A)$ corresponds to A being a generalized 2-stage matrix. Indeed, the algorithms presented by [30] are based on reducing the problem to the n-fold and 2-stage cases and applying the algorithms reviewed above.

2.2 Unified Approach and Treedepth

Denote by $\langle A, f, \mathbf{b}, \mathbf{l}, \mathbf{u}\rangle$ the binary encoding length of an IP instance. (We define the encoding length of f to be the length of f_{gap}, which is the difference between the maximum and minimum values of f on the domain.) The function f is given by an oracle. A glaring question left open by Ganian and Ordyniak [42] is that of the complexity of IP parameterized by $\|A\|_\infty$ and $\mathrm{td}_P(A)$ or $\mathrm{td}_D(A)$. In [73], we have fully settled this with the following result:

▶ **Theorem 1.** *There exists a computable function g such that IP can be solved in time*

$$g(\|A\|_\infty, d)\mathrm{poly}(n, L), \qquad \textit{where } d := \min\{\mathrm{td}_P(A), \mathrm{td}_D(A)\} \textit{ and } L := \langle A, f, \mathbf{b}, \mathbf{l}, \mathbf{u}\rangle \ .$$

Note that for this algorithm to be fast it suffices if at least one of $\mathrm{td}_P(A)$ and $\mathrm{td}_D(A)$ is small. Also, all the presented results hold for IP whose constraints are given in the inequality form $A\mathbf{x} \leq \mathbf{b}$: introducing slack variables leads to IP in standard form with a constraint matrix $A_I := (A\ I)$, with $\min\{\mathrm{td}_P(A_I), \mathrm{td}_D(A_I)\} \leq \min\{\mathrm{td}_P(A) + 1, \mathrm{td}_D(A)\}$.

Theorem 1 is qualitatively optimal in the following sense. Already for $\|A\|_\infty = 1$ or $d = 1$ ILP is NP-hard: by the natural reduction from SAT in the former case, and similarly by the natural modeling of Knapsack in the latter. Moreover, the two arguably most important tractable classes of IP are formed by instances whose constraint matrix is either totally unimodular or has small number n of columns, yet our results are incomparable with either: the class of totally unimodular matrices might have large d, but has $\|A\|_\infty = 1$, and the matrices considered here have variable n.

Furthermore, treedepth cannot be replaced with the more permissive notion of treewidth, since IP is NP-hard already when $\min\{\mathrm{tw}_P(A), \mathrm{tw}_D(A)\} = 2$ and $a = 2$ [33]. Second, the parameterization cannot be relaxed by removing the parameter $\|A\|_\infty$: IP is para-NP-hard parameterized by $\mathrm{td}_P(A)$ [30, Thm 21] and strongly W[1]-hard parameterized by $\mathrm{td}_D(A)$ [72, Thm 5] alone. Third, the requirement that f is separable convex cannot be relaxed since IP with a non-separable convex or a separable concave function are NP-hard even for small values of our parameters [33]. Let us now sketch our approach towards proving Theorem 1.

Treedepth is Equivalent to Block Structure. The first ingredient is showing that, assuming small $\|A\|_\infty$, matrices with small primal treedepth are essentially equivalent to multi-stage Wmatrices, and similarly that matrices with small dual treedepth are essentially equivalent to tree-fold matrices. Thus, in a certain sense, it suffices to restrict our attention to the (more rigidly structured) classes of multi-stage stochastic and tree-fold matrices.

Graver-best Augmentation. The second ingredient is an abstraction of an iterative augmentation procedure used in [52] and elsewhere: a *Graver-best oracle* for a matrix A is one which, when queried on an IP instance $(A, f, \mathbf{b}, \mathbf{l}, \mathbf{u})$ and a feasible solution $\mathbf{x}$, returns a feasible step $\mathbf{h}$ which is at least as good as any feasible step $\lambda\mathbf{g}$ where $\mathbf{g} \in \mathcal{G}(A)$ and $\lambda \in \mathbb{N}$. Given such an oracle, we can solve IP quickly by iteratively querying it on the current solution and adding the returned step to the solution.

Norm Bounds and Local Search. We identify that constructing efficient Graver-best oracles requires two components. The first are bounds on the norms of Graver basis elements. Denote by $g_\infty(A) = \max_{\mathbf{g}\in\mathcal{G}(A)} \|\mathbf{g}\|_\infty$, and similarly $g_1(A) = \max_{\mathbf{g}\in\mathcal{G}(A)} \|\mathbf{g}\|_1$. We show that the structural result of Aschenbrenner and Hemmecke [4] can be interpreted as saying that $g_\infty(A)$ is bounded by a function of $\|A\|_\infty$ and $\mathrm{td}_P(A)$, and similarly that the work of Chen and Marx [14] implies that $g_1(A)$ is bounded by a function of $\|A\|_\infty$ and $\mathrm{td}_D(A)$.

Given these norm bounds, it becomes apparent that computing Graver-best steps boils down to solving a local search problem. This second component can be obtained by a relatively simple branching algorithm in the case of $\mathrm{td}_P(A)$, and by a more involved Knapsack-style dynamic programming algorithm in the case of $\mathrm{td}_D(A)$.

2.3 Strongly Polynomial Framework

A fundamental question regarding problems involving large numbers is whether there exists an algorithm whose number of arithmetic operations does not depend on the length of the numbers involved; recall that if this number is polynomial, this is a *strongly polynomial algorithm* [93]. For example, the ellipsoid method or the interior-point method which solve LP take time which does depend on the encoding length, and the existence of a strongly polynomial algorithm for LP remains a major open problem. Until 2018, the only strongly polynomial ILP algorithms existed for totally unimodular ILP [56], bimodular ILP [3], so-called binet ILP [2], and n-fold ILP with constant block dimensions [25]. All remaining results, such as Lenstra's famous algorithm or the FPT algorithm for n-fold IP of Hemmecke et al. [52], are not strongly polynomial.

A second contribution of [73] besides Theorem 1 is the development of an algorithmic framework among others suitable for obtaining strongly polynomial algorithms, and as a consequence of Theorem 1 we show a strongly polynomial algorithm for ILP in the same parameter regime:

▶ **Theorem 2.** *There exists a computable function g such that ILP can be solved with an algorithm whose number of arithmetic operations is bounded by*

$$g(\|A\|_\infty, d)\text{poly}(n), \qquad \text{where } d := \min\{\text{td}_P(A), \text{td}_D(A)\} \ .$$

The proof of Theorem 2 proceeds in four steps:

1. **LP relaxation:** the LP relaxation can be solved in time $\text{poly}(n \cdot \langle A \rangle)$ by Tardos' algorithm [93], obtaining a fractional optimum $\mathbf{x}^*$.
2. **Proximity:** the integer optimum $\mathbf{z}^*$ which we seek is at an ℓ_∞-distance at most $\mathcal{O}(g_\infty(A) \cdot n)$ from $\mathbf{x}^*$, thus we may reduce the lower and upper bounds $\mathbf{l}, \mathbf{u}$ to some $\mathbf{l}', \mathbf{u}'$ which are bounded by $\mathcal{O}(g_\infty(A) \cdot n)$, and subsequently also reduce the right hand side $\mathbf{b}$ to some $\mathbf{b}'$ which is bounded by $g_\infty(A)\text{poly}(n)$.
3. **Objective reduction:** since we now know that the integer optimum $\mathbf{z}^*$ lies in a "small" box $[\mathbf{l}', \mathbf{u}']$, we can apply the coefficient reduction technique of Frank and Tardos [40] to obtain an equivalent objective $\mathbf{w}'$ with $\log \|\mathbf{w}'\|_\infty = \text{poly}(g_\infty(A) \cdot n)$.
4. **Convergence:** because the length of all numbers on input is now polynomial in n, the algorithm of Theorem 1 runs in strongly polynomial time.

A nice and somewhat surprising property of this approach is that, since f is given by a comparison oracle and since step 3 reduces it to an equivalent function, this step is actually irrelevant and we may proceed to step 4 directly. Thus, the algorithm boils down to solving the LP relaxation, reducing the lower and upper bounds and right hand side in a straightforward manner, and running the algorithm of Theorem 1. The fact that step 3 is unnecessary is in fact substantial, because while the algorithm of [40] is polynomial, the degree of this polynomial is quite large and would dominate the complexity of the algorithm overall. We have expanded on these ideas in [32] by showing that if the equivalent objective need not be computed, then stronger bounds on $\|\mathbf{w}'\|_\infty$ can be obtained, leading to better bounds on the algorithm itself. Furthermore, we have shown reducibility upper bounds on not just linear but also separable convex functions, and also almost matching lower bounds.

2.4 Simplified and Self-contained

The main contribution of [31] over [73] is that it presents a streamlined and self-contained version of the proof of Theorem 1. With the benefit of hindsight, we have replaced Graver-best augmentation with a cheaper so-called halfling augmentation, and we have derived all results (norm bounds, local search programs) directly in the most general setting of bounded treedepth. The whole paper is only 16 pages long including references, and is currently perhaps the best entry point into the area for any newcomer, including graduate and even undergraduate students. Still, we are (slowly) working on developing lecture notes, which will be even more approachable.

3 High-multiplicity n-fold IP

Much research following [73] has focused on reducing the run-time dependency on the number of bricks n, making it almost linear in the case of optimizing a linear objective [19, 20]. Our interest now is on n-fold IP models of applications where many bricks are of the same *type*, that is, they share the same bounds, right-hand side, and objective function. For those applications, it is natural to encode an n-fold IP instance *succinctly* by describing each brick type by its constraint matrix, bounds, right-hand side, and objective function, and giving a vector of brick multiplicities. When the number of brick types τ is much smaller than

the number n of bricks, e.g., if $n \approx 2^\tau$, this succinct instance is (much) smaller than the "explicit" encoding of n-fold IP, and an algorithm running in time polynomial in the size of the succinct instance may be (much) faster than current algorithms. We call the n-fold IP where the instance is given succinctly the *huge* (or *high multiplicity*) *n-fold IP* problem, and in [66] we present a fast algorithm for it:

▶ **Theorem 3.** *Huge n-fold IP with any separable convex objective can be solved in time*

$$(\|A\|_\infty rs)^{\mathcal{O}(r^2s+rs^2)}\text{poly}(\tau, t, \log \|\mathbf{l}, \mathbf{u}, \mathbf{b}, n, f_{\text{gap}}\|_\infty) \ .$$

A natural application of Theorem 3 are *high-multiplicity scheduling problems* [18, 55, 89].

Proof Ideas. To solve a high-multiplicity problem, one needs a succinct description of solutions. The fundamental and influential notion of Configuration IP (ConfIP) introduced by Gilmore and Gomory [45] describes a solution (e.g., a schedule) by a list of pairs "(machine schedule s, multiplicity μ of machines with schedule s)". The linear relaxation of ConfIP, called the Configuration LP (ConfLP), can often be solved efficiently, and is known to provide solutions of strikingly high quality in practice [96]; for example, the optimum of the ConfLP for Bin Packing is conjectured to have value x such that an optimal integer packing uses $\leq \lceil x \rceil + 1$ bins [92]. However, surprisingly little is known *in general* about the structure of solutions of ConfIP and ConfLP, and how they relate to each other.

We define the Configuration IP and LP of an n-fold IP instance, and show how to solve the ConfLP quickly using the property that the ConfLP and ConfIP have polynomial encoding length even for huge n-fold IP. Our main technical contribution is a proximity theorem about n-fold IP, showing that a solution of its relaxation corresponding to the ConfLP optimum is very close to the integer optimum. Thus, the algorithm of Theorem 3 proceeds in three steps:

1. It solves the ConfLP via the standard approach of considering the dual and its separation problem, which in this case turns out to be an efficiently solvable IP,
2. It uses the novel proximity theorem to construct a "residual" n'-fold instance with n' upperbounded by $(\|A\|_\infty rs)^{\mathcal{O}(rs)}$, and
3. it solves the residual instance by an existing n-fold IP algorithm such as Theorem 1.

The proximity theorem provides insight into the fundamental notion of Configuration LP. Intuitively, it means the following: there is an integer optimum of the Configuration LP which agrees with the continuous optimum on "most" configurations, and where it differs, it only deviates to configurations which are not "too far". In other words, an optimum of the Configuration LP which only uses few configurations implies the existence of an integer optimum which puts "most" weight on these configurations, and puts the remaining weight in "small" balls around these configurations. (Note that a continuous optimum using only few configurations can be found efficiently whenever *some* optimum can be found at all.)

We highlight the proximity theorem also because the strongly near-linear algorithm of Cslovjecsek et al. [19] uses a special case of our relaxation of an n-fold IP in order to obtain a similarly tight proximity result as we do, however, their result only holds for linear objectives, and this seems inherent. Because our proximity theorem holds also for separable convex objectives, it allowed us in a subsequent work [68] to show efficient preprocessing algorithms (kernels) for certain scheduling problems, including those whose models involve separable convex objectives.

4 Mixed ILP and IP

Consider again the celebrated result of Lenstra [75] that ILP is FPT parameterized by the number of variables n. Lenstra's brilliant (and short) paper also shows an extension of this result to Mixed ILP where both integer and non-integer variables are allowed:

$$\min\{\mathbf{wx} \mid A\mathbf{x} = \mathbf{b}, \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}, \mathbf{x} \in \mathbb{Z}^z \times \mathbb{R}^q\}\ , \tag{MILP}$$

with $A \in \mathbb{Z}^{m\times(z+q)}$, $\mathbf{l}, \mathbf{u}, \mathbf{w} \in \mathbb{Z}^{z+q}$ and $\mathbf{b} \in \mathbb{Z}^m$. Specifically, Lenstra showed that MILP is FPT parameterized by the number of integer variables z, so the number of continuous variables q is allowed to be variable. MILP is a prominent modelling tool widely used in practice. For example, Bixby [8] says in his famous analysis of LP solver speed-ups, *"[I]nteger programming, and most particularly the mixed-integer variant, is the dominant application of linear programming in practice."* Given the fact that Lenstra's result extends to the mixed case, it is natural to ask whether the FPT algorithm of Theorem 1 can too be extended to the MILP case. This is the subject of [10]. In [9], we explore the more general setting of Mixed IP with separable convex objectives.

4.1 Linear Optimization

The main result of [10] is the following:

▶ **Theorem 4.** *There exists a computable function g such that MILP can be solved in time*

$$g(\|A\|_\infty, d)\text{poly}(n, L), \qquad \textit{where } d := \min\{\text{td}_P(A), \text{td}_D(A)\},\ \textit{and } L := \langle A, f, \mathbf{b}, \mathbf{l}, \mathbf{u}\rangle\ .$$

By the techniques of Theorem 2, the algorithm of Theorem 4 can be made strongly FPT.

Proof Ideas. The proof goes by reducing an MILP instance to an ILP instance whose parameters do not increase too much, and then applying the existing algorithms (e.g., Theorem 2) for ILP. The key is considering the *fractionality* of an MILP instance, which is the minimum of the maxima of the denominators in optimal solutions. For example, it is well-known that the natural LP for the Vertex Cover problem has half-integral optima, that is, there exists an optimum with all values in $\{0, \frac{1}{2}, 1\}$.

The usual way to go about proving fractionality bounds is via Cramer's rule and a sufficiently good bound on the determinant. However, the determinants can grow large even for matrices of very benign structure. (For example, the determinant of the $n \times n$ matrix $2I$ is 2^n.) Instead, we analyse carefully the structure of the inverse of the appearing invertible sub-matrices, allowing us to show that an MILP instance with a constraint matrix A has an optimal solution $\mathbf{x}$ whose largest denominator is bounded by $(\|A\|_\infty)^{d!}(d!)^{d!/2}$, where $d = \min\{\text{td}_P(A), \text{td}_D(A)\}$. Intuitively, this means that an LP or MILP with small treedepth and coefficients has vertices which are not "too fractional", that is, its vertices lie on the superlattice $\frac{1}{s}\mathbb{Z}^{z+q}$ (more precisely $\mathbb{Z}^z \times \frac{1}{s}\mathbb{Z}^q$) with s not too large. To the best of our knowledge, this is the first time an algorithm for ILP has been lifted to MILP in this way.

We also show the limits of this approach by lower-bounding the fractionality of inverses for other classes of matrices where efficient ILP algorithms are known.

A fascinating connection has emerged between fractionality bounds, and bounds on the norms of the circuits of a matrix A, which are a subset of the Graver basis. Ekbatani et al. [35] show that if, for any integral $\mathbf{b}$, the polytope $\{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\}$ is not too fractional, then the norm of the circuits of A is not too large, and vice versa. Since the circuits are a subset of $\mathcal{G}(A)$, and we already have good bounds on $\mathcal{G}(A)$ in the considered regime, the

result of Ekbatani et al. implies a fractionality bound; however, our approach yields a tighter bound. The connection can be viewed from the other side as well: our upper bound on the fractionality can be used to show a circuit norm upper bound.

4.2 Separable Convex Optimization

Lenstra's algorithm [75] can also be extended, e.g. using [49, Theorem 6.7.9], to the setting of arbitrary convex objective functions, and, by the same arguments as with the step from ILP to MILP, it can be shown that MIP, which is the problem

$$\min\{f(\mathbf{x}) \mid A\mathbf{x} = \mathbf{b}, \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}, \mathbf{x} \in \mathbb{Z}^z \times \mathbb{R}^q\}, \tag{MIP}$$

with f convex is also FPT parameterized by z. Giving attention to objective functions beyond linear is well justified: Bertsimas et al. note in their spectacular work [6] on the notorious subset selection problem in statistical learning that, over the past decades, algorithmic and hardware advances have elevated *convex* (and therefore, non-linear) mixed-integer optimization to a comparable level of relevance in applications. Given the encouraging results of [10], one may ask whether the FPT algorithm of Theorem 1 can be extended to the mixed-integer, separable convex case. The *a priori* intuition leans positive: in 1990, Hochbaum and Shanthikumar [56] have published an influential paper titled "Convex separable optimization is not much harder than linear optimization" which shows that this is true in the case of IP with a totally unimodular A. Similarly, Chubanov's algorithm [16] reduces continuous separable convex optimization to a polynomial number of linear optimization problems.

In [9], we go directly against this intuition, in fact, we show that the mixed integer separable convex case exhibits unexpected behavior both when compared to the fully integer cases, and to the linear cases. We begin with the regime of few rows and small coefficients:

▶ **Theorem 5.** *MIP can be solved in time* $(m\|A\|_\infty)^{\mathcal{O}(m^2)} \cdot \mathcal{R}$*, where* $\mathcal{R}$ *is the time needed to solve the continuous relaxation of any MIP with the constraint matrix* A*.*

This improves the current state-of-the-art, double-exponential bound for MIP with few rows and small coefficients to single-exponential, even when the target function is non-linear. Until now, the best way to solve a MIP with few rows and small coefficients would be to remove duplicate columns from A in a preprocessing step, and then use Lenstra's algorithm [75].

The structural result behind Theorem 5 is an upper bound on the elements of the mixed Graver basis of A. We show this bound using a "packing lemma", powered by an old result from additive combinatorics [84] recently highlighted by Paat et al. [85]. Using this bound we show that, for any continuous or integer optimum, there is a mixed-integer optimum which may require several mixed-integer Graver steps to get to, but only one of those steps will be non-zero in the integer part. Thus, an integer or continuous optimum is always "almost correct" in the integer part. Once the integer part is correctly assigned, we get a purely continuous residual problem, which is polynomial-time solvable.

Set $\mathbb{X} = \mathbb{Z}^z \times \mathbb{R}^q$. Consider the problem where we allow the bounds $\mathbf{l}, \mathbf{u} \in \mathbb{X}$ and right-hand side $\mathbf{b} \in \mathbb{R}^m$ to be fractional:

$$\min\{\mathbf{w}\mathbf{x} : E\mathbf{x} = \mathbf{b}, \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}, \mathbf{x} \in \mathbb{X}\}. \tag{MILP$_{\text{frac}}$}$$

Already deciding feasibility of this variant is NP-hard for totally unimodular matrices [17]. In a way, MILP$_{\text{frac}}$ can be seen as a special case of MIP (even with integer $\mathbf{b}, \mathbf{l}, \mathbf{u}$), because a separable convex objective f can be used to model non-integer lower bounds and right-hand sides. For the case of 2-stage constraints, we give an XP algorithm:

▶ **Theorem 6.** *The problem $MILP_{frac}$ where A is a 2-stage matrix, can be solved in time $g(r, s, \|A\|_\infty) \cdot n^r$, for some computable function g.*

This result turns out to be likely optimal, as we show that MIP with integral data is W[1]-hard when A is a 2-stage matrix with blocks of size bounded by a parameter and $\|A\|_\infty = 1$ already for linear objective functions. Moreover, we show that Theorem 6 cannot be extended to the related case of n-fold constraint structure: $\text{MILP}_{\text{frac}}$ with integral bounds but fractional right hand sides is NP-hard when A is an n-fold matrix with blocks of constant dimensions and $\|A\|_\infty = 1$.

5 Beyond Small Coefficients and Graver Bases – Coming Full Circle

Intuitively, 2-stage matrices have few "global" columns, and n-fold matrices have few "global" rows, whose deletion makes the matrix block-diagonal, respectively. The combination of both is a 4-block n-fold matrix, which has few rows and few columns which are "global". Hemmecke et al. [51] have introduced this class and shown an XP algorithm for IP with such matrices, when parameterizing by the block dimensions and $\|A\|_\infty$. The arguably most important question about block-structured IP is whether this algorithm is qualitatively optimal, that is, whether 4-block n-fold IP is W[1]-hard or FPT.

By employing standard encoding tricks, any 2-stage stochastic IP and any n-fold IP instance with small block sizes but containing entries bounded by $\text{poly}(n)$ in its "global" parts can be reduced to a 4-block n-fold IP instance with small coefficients and small block sizes. Thus, if 4-block n-fold IP is FPT, then at least the aforementioned two generalizations of 2-stage and n-fold IP are FPT as well.

In [21], we prove that both the feasibility of 2-stage ILP and the optimization of uniform n-fold ILP (i.e., with all global blocks identical) are FPT when parameterized by the dimensions of the blocks and the maximum absolute value of any entry appearing in the diagonal blocks. That is, we allow the entries of the global blocks to be arbitrarily large, and in the case of n-fold programs, we require that all global blocks are equal:

▶ **Theorem 7.** *Let L be the encoding length of the ILP instances mentioned below.*

1. *The feasibility of a generalized (i.e., blocks may differ) 2-stage ILP can be solved in time $g(k, \max_i \|A_2^i\|_\infty) \cdot L$ for a computable function g, where k is the largest number of columns of any block, and A_2^i, $i \in [n]$, are the diagonal blocks.*
2. *n-fold ILP with all global blocks identical can be solved in time $g(k, \max \|A_2^i\|_\infty) \cdot \text{poly}(L)$ for a computable function g, where k is the largest number of rows and columns of any block, and A_2^i, $i \in [n]$, are the diagonal blocks.*

The uniformity condition (all global blocks identical) for n-fold ILP is necessary (unless P = NP), as one can reduce Subset Sum to a feasibility n-fold ILP with $k = 2$ and the diagonal blocks being $\{0, 1\}$-matrices. Regarding 2-stage ILP, Eisenbrand and Rothvoss have recently shown an elegant extension to the optimization variant [34].

Proof Ideas. The two parts of Theorem 7 each rely on different techniques, and, not surprisingly, they significantly depart from the approach through Graver bases. In both cases, the problem is ultimately reduced to (mixed) integer programming with a bounded number of (integral) variables, which is then solved using Lenstra's algorithm [75] (or any of its newer strengthenings [62, 90]), and this allows it to cope with large entries on input.

This connection to fixed-dimension ILP is fascinating: we have come full circle, from the old tractable class of small dimension programs, through the newer and seemingly unrelated class of block-structured programs, back to programs with small dimension. Still, the present

techniques do not seem to extend to the case of *separable convex* objectives. We believe that extending the results above to separable convex objectives, or ruling out that such an extension exists, is an important open problem.

The first part of Theorem 7 is based on a new structural result about integer cones, which allows us to process the (many) input constraints and create a fixed-dimension ILP describing the feasible set of global variables. Once those are found and fixed, the problem decomposes into independent and small subproblems.

The second part of Theorem 7 relies on the following Brick Decomposition Lemma: for each brick $i \in [n]$, the right hand side $\mathbf{b}^i$ can be decomposed into two vectors $(\mathbf{b}^i)'$ and $(\mathbf{b}^i)''$ such that $\mathbf{b}^i = (\mathbf{b}^i)' + (\mathbf{b}^i)''$, $(\mathbf{b}^i)', (\mathbf{b}^i)''$ are in the same orthant as $\mathbf{b}^i$ and below it, and, most importantly, they have the property that every solution $\mathbf{x}^i$ satisfying $A\mathbf{x}^i = \mathbf{b}^i$ can be decomposed into $\mathbf{x}^i = (\mathbf{x}^i)' + (\mathbf{x}^i)''$ such that $A(\mathbf{x}^i)' = (\mathbf{b}^i)'$ and $A(\mathbf{x}^i)'' = (\mathbf{b}^i)''$. Iteratively applying this decomposition, we can reduce the problem to a high-multiplicity n-fold ILP with few brick types, since eventually each right hand side becomes small. Note that because of the large coefficients in the global constraints, this instance is not solvable by Theorem 3. Still, this instance can be massaged to a form solvable by Lenstra's algorithm.

Recall the n-fold model of the makespan minimization on uniformly related machines ($Q||C_{\max}$) problem introduced in Section 1.2. This is a program with many bricks which only differ by their right hand sides. Applying the decomposition technique can then be interpreted as replacing one fast machine with two slower machines, and the result of iterating this process is that there are "few" different machines, and each has a "small" capacity. This seems to be the gist of an argument carried out by Brinkop and Jansen [12], so their result can be seen as a particular instance of the decomposition insight above. Also, the fact that $Q||C_{\max}$ is FPT parameterized by $p_{\max}$ can now either be shown by a direct application to n-fold ILP (as we have done in [67]), or by the argument above and then solving the resulting fixed-dimension ILP instance using, e.g., the algorithm of Reis and Rothvoss [90]. This second approach gives a worse complexity bound, but provides a useful new perspective.

References

1 Sibel A. Alumur and Bahar Yetis Kara. Network hub location problems: The state of the art. *European Journal of Operational Research*, 190(1):1–21, 2008. doi:10.1016/J.EJOR.2007.06.008.

2 Gautam Appa, Balázs Kotnyek, Konstantinos Papalamprou, and Leonidas Pitsoulis. Optimization with binet matrices. *Operations research letters*, 35(3):345–352, 2007. doi:10.1016/J.ORL.2006.04.003.

3 Stephan Artmann, Robert Weismantel, and Rico Zenklusen. A strongly polynomial algorithm for bimodular integer linear programming. In *STOC 2017*, pages 1206–1219. ACM, 2017. doi:10.1145/3055399.3055473.

4 Matthias Aschenbrenner and Raymond Hemmecke. Finiteness theorems in stochastic integer programming. *Foundations of Computational Mathematics*, 7(2):183–227, 2007. doi:10.1007/S10208-005-0174-1.

5 Jacques F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Comput. Manag. Sci.*, 2(1):3–19, 2005. doi:10.1007/S10287-004-0020-Y.

6 Dimitris Bertsimas, Angela King, and Rahul Mazumder. Best subset selection via a modern optimization lens. *The Annals of Statistics*, 44(2):813–852, 2016. URL: http://www.jstor.org/stable/43818629.

7 John R. Birge and François Louveaux. *Introduction to stochastic programming*. Springer-Verlag, New York, 1997.

8 Robert E Bixby. Solving real-world linear programs: A decade and more of progress. *Operations research*, 50(1):3–15, 2002. doi:10.1287/OPRE.50.1.3.17780.

9 Cornelius Brand, Martin Koutecký, Alexandra Lassota, and Sebastian Ordyniak. Separable Convex Mixed-Integer Optimization: Improved Algorithms and Lower Bounds. In *ESA 2024*, volume 308 of *LIPIcs*, pages 32:1–32:18, Dagstuhl, Germany, 2024. doi:10.4230/LIPIcs.ESA.2024.32.

10 Cornelius Brand, Martin Koutecký, and Sebastian Ordyniak. Parameterized algorithms for MILPs with small treedepth. In *AAAI 2021*, pages 12249–12257. AAAI Press, 2021. doi:10.1609/AAAI.V35I14.17454.

11 Marcin Briański, Martin Koutecký, Daniel Kráľ, Kristýna Pekárková, and Felix Schröder. Characterization of matrices with bounded graver bases and depth parameters and applications to integer programming. *Mathematical Programming*, pages 1–35, 2024. doi:10.1007/s10107-023-02048-x.

12 Hauke Brinkop and Klaus Jansen. High multiplicity scheduling on uniform machines in fpt-time. *CoRR*, abs/2203.01741, 2022. doi:10.48550/arXiv.2203.01741.

13 Timothy F. N. Chan, Jacob W. Cooper, Martin Koutecký, Daniel Kráľ, and Kristýna Pekárková. Matrices of optimal tree-depth and a row-invariant parameterized algorithm for integer programming. *SIAM J. Comput.*, 51(3):664–700, 2022. doi:10.1137/20m1353502.

14 Lin Chen and Dániel Marx. Covering a tree with rooted subtrees–parameterized and approximation algorithms. In *SODA 2018*, pages 2801–2820. SIAM, 2018. doi:10.1137/1.9781611975031.178.

15 Lin Chen, Dániel Marx, Deshi Ye, and Guochuan Zhang. Parameterized and approximation results for scheduling with a low rank processing time matrix. In *STACS 2017*, volume 66 of *LIPIcs*, pages 22:1–22:14, 2017. doi:10.4230/LIPICS.STACS.2017.22.

16 Sergei Chubanov. A polynomial-time descent method for separable convex optimization problems with linear constraints. *SIAM Journal on Optimization*, 26(1):856–889, 2016. doi:10.1137/14098524X.

17 Michele Conforti, Marco Di Summa, Friedrich Eisenbrand, and Laurence A. Wolsey. Network formulations of mixed-integer programs. *Math. Oper. Res.*, 34(1):194–209, 2009. doi:10.1287/moor.1080.0354.

18 Stavros S. Cosmadakis and Christos H. Papadimitriou. The traveling salesman problem with many visits to few cities. *SIAM J. Comput.*, 13(1):99–108, 1984. doi:10.1137/0213007.

19 Jana Cslovjecsek, Friedrich Eisenbrand, Christoph Hunkenschröder, Lars Rohwedder, and Robert Weismantel. Block-structured integer and linear programming in strongly polynomial and near linear time. In *SODA 2021*, pages 1666–1681. SIAM, 2021. doi:10.1137/1.9781611976465.101.

20 Jana Cslovjecsek, Friedrich Eisenbrand, Michał Pilipczuk, Moritz Venzin, and Robert Weismantel. Efficient sequential and parallel algorithms for multistage stochastic integer programming using proximity. In *ESA 2021*, volume 204 of *LIPIcs*, pages 33:1–33:14, 2021. doi:10.4230/LIPIcs.ESA.2021.33.

21 Jana Cslovjecsek, Martin Koutecký, Alexandra Lassota, Michał Pilipczuk, and Adam Polak. Parameterized algorithms for block-structured integer programs with large entries. In *SODA 2024*, pages 740–751. SIAM, 2024. doi:10.1137/1.9781611977912.29.

22 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

23 George B Dantzig. Application of the simplex method to a transportation problem. *Activity Analysis and Production and Allocation*, 1951.

24 George B Dantzig and Philip Wolfe. Decomposition principle for linear programs. *Operations research*, 8(1):101–111, 1960.

25 Jesús A. De Loera, Raymond Hemmecke, and Jon Lee. On augmentation algorithms for linear and integer-linear programming: From Edmonds–Karp to Bland and beyond. *SIAM Journal on Optimization*, 25(4):2494–2511, 2015. doi:10.1137/151002915.

26 Jesús A. De Loera, Raymond Hemmecke, Shmuel Onn, and Robert Weismantel. N-fold integer programming. *Discrete Optimization*, 5(2):231–241, 2008. doi:10.1016/J.DISOPT.2006.06.006.

27 Jesús A. De Loera and Shmuel Onn. The complexity of three-way statistical tables. *SIAM J. Comput*, 33(4):819–836, 2004. doi:10.1137/S0097539702403803.

28 Jesús A. De Loera and Shmuel Onn. All linear and integer programs are slim 3-way transportation programs. *SIAM Journal on Optimization*, 17(3):806–821, 2006. doi:10.1137/040610623.

29 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.

30 Pavel Dvořák, Eduard Eiben, Robert Ganian, Dušan Knop, and Sebastian Ordyniak. The complexity landscape of decompositional parameters for ILP: programs with few global variables and constraints. *Artif. Intell.*, 300:103561, 2021. doi:10.1016/j.artint.2021.103561.

31 Friedrich Eisenbrand, Christoph Hunkenschröder, Kim-Manuel Klein, Martin Koutecký, Asaf Levin, and Shmuel Onn. Sparse integer programming is fixed-parameter tractable. *Mathematics of Operations Research*, 0(0):null, 0. doi:10.1287/moor.2023.0162.

32 Friedrich Eisenbrand, Christoph Hunkenschröder, Kim-Manuel Klein, Martin Koutecký, Asaf Levin, and Shmuel Onn. Reducibility bounds of objective functions over the integers. *Oper. Res. Lett.*, 51(6):595–598, 2023. doi:10.1016/J.ORL.2023.10.001.

33 Friedrich Eisenbrand, Christoph Hunkenschröder, Kim-Manuel Klein, Martin Koutecký, Asaf Levin, and Shmuel Onn. An algorithmic theory of integer programming, 2019. arXiv:1904.01361.

34 Friedrich Eisenbrand and Thomas Rothvoss. A parameterized linear formulation of the integer hull. *CoRR*, abs/2501.02347, 2025. doi:10.48550/arXiv.2501.02347.

35 Farbod Ekbatani, Bento Natura, and László A. Végh. Circuit imbalance measures and linear programming. *CoRR*, abs/2108.03616, 2021. arXiv:2108.03616.

36 Piotr Faliszewski, Rica Gonen, Martin Koutecký, and Nimrod Talmon. Opinion diffusion and campaigning on society graphs. *J. Log. Comput.*, 32(6):1162–1194, 2022. doi:10.1093/logcom/exac014.

37 Stephen E Fienberg and Alessandro Rinaldo. Three centuries of categorical data analysis: Log-linear models and maximum likelihood estimation. *Journal of Statistical Planning and Inference*, 137(11):3430–3445, 2007.

38 Samuel Fiorini, Gwenaël Joret, Stefan Weltge, and Yelena Yuditsky. Integer programs with bounded subdeterminants and two nonzeros per row. In *FOCS 2021*, pages 13–24. IEEE, 2021. doi:10.1109/FOCS52979.2021.00011.

39 Christodoulos A. Floudas and Xiaoxia Lin. Mixed integer linear programming in process scheduling: Modeling, algorithms, and applications. *Ann. Oper. Res.*, 139(1):131–162, 2005. doi:10.1007/S10479-005-3446-X.

40 András Frank and Éva Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Combinatorica*, 7(1):49–65, 1987. doi:10.1007/BF02579200.

41 Eugene C. Freuder. Complexity of K-tree structured constraint satisfaction problems. In *Proceedings of the 8th National Conference on Artificial Intelligence*, pages 4–9, 1990. URL: http://www.aaai.org/Library/AAAI/1990/aaai90-001.php.

42 Robert Ganian and Sebastian Ordyniak. The complexity landscape of decompositional parameters for ILP. *Artificial Intelligence*, 2018.

43 Robert Ganian, Sebastian Ordyniak, and M. S. Ramanujan. Going beyond primal treewidth for (M)ILP. In *AAAI*, pages 815–821, 2017. doi:10.1609/AAAI.V31I1.10644.

44 Tomáš Gavenčiak, Martin Koutecký, and Dušan Knop. Integer programming in parameterized complexity: Five miniatures. *Discret. Optim.*, 44(Part):100596, 2022. doi:10.1016/j.disopt.2020.100596.

45 Paul C Gilmore and Ralph E Gomory. A linear programming approach to the cutting-stock problem. *Operations research*, 9(6):849–859, 1961.

46 Paul C Gilmore and Ralph E Gomory. A linear programming approach to the cutting stock problem – part ii. *Operations research*, 11(6):863–888, 1963.

47 Fred W. Glover. Tabu search - part II. *INFORMS J. Comput.*, 2(1):4–32, 1990. doi:10.1287/IJOC.2.1.4.

48 Dmitry V. Gribanov, Dmitry S. Malyshev, and Ivan A. Shumilov. On a simple connection between Δ-modular ILP and lp, and a new bound on the number of integer vertices. *Oper. Res. Forum*, 5(2):32, 2024. doi:10.1007/S43069-024-00310-2.

49 Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric algorithms and combinatorial optimization*, volume 2 of *Algorithms and Combinatorics*. Springer-Verlag, Berlin, second edition, 1993.

50 Claire Hanen and Alix Munier Kordon. Fixed-parameter tractability of scheduling dependent typed tasks subject to release times and deadlines. *J. Sched.*, 27(2):119–133, 2024. doi:10.1007/S10951-023-00788-4.

51 Raymond Hemmecke, Matthias Köppe, and Robert Weismantel. Graver basis and proximity techniques for block-structured separable convex integer minimization problems. *Mathematical Programming*, 145(1-2, Ser. A):1–18, 2014. doi:10.1007/S10107-013-0638-Z.

52 Raymond Hemmecke, Shmuel Onn, and Lyubov Romanchuk. N-fold integer programming in cubic time. *Mathematical Programming*, pages 1–17, 2013.

53 Raymond Hemmecke and Rüdiger Schultz. Decomposition of test sets in stochastic integer programming. *Mathematical Programming*, 94:323–341, 2003. doi:10.1007/S10107-002-0322-1.

54 Frank L Hitchcock. The distribution of a product from several sources to numerous localities. *Journal of mathematics and physics*, 20(1-4):224–230, 1941.

55 Dorit S. Hochbaum and Ron Shamir. Strongly polynomial algorithms for the high multiplicity scheduling problem. *Oper. Res.*, 39(4):648–653, 1991. doi:10.1287/OPRE.39.4.648.

56 Dorit S. Hochbaum and J. George Shanthikumar. Convex separable optimization is not much harder than linear optimization. *Journal of the ACM*, 37(4):843–862, 1990. doi:10.1145/96559.96597.

57 Alan J Hoffman and Joseph B Kruskal. Integral boundary points of convex polyhedra. *Linear inequalities and related systems*, pages 223–246, 1956.

58 Serkan Hoşten and Seth Sullivant. A finiteness theorem for markov bases of hierarchical models. *J. Comb. Theory A*, 114(2):311–321, 2007. doi:10.1016/J.JCTA.2006.06.001.

59 Bart M. P. Jansen and Stefan Kratsch. A structural approach to kernels for ILPs: Treewidth and total unimodularity. In *ESA*, volume 9294 of *Lecture Notes in Computer Science*, pages 779–791, 2015. doi:10.1007/978-3-662-48350-3_65.

60 Klaus Jansen, Kim-Manuel Klein, Marten Maack, and Malin Rau. Empowering the configuration-IP: new PTAS results for scheduling with setup times. *Math. Program.*, 195(1):367–401, 2022. doi:10.1007/S10107-021-01694-3.

61 Peter Kall and Stein W. Wallace. *Stochastic Programming*. Wiley, Chichester etc., 1994.

62 Ravi Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of Operations Research*, 12(3):415–440, August 1987. doi:10.1287/MOOR.12.3.415.

63 Leonid V Kantorovich. On the translocation of masses. In *Dokl. Akad. Nauk. USSR (NS)*, volume 37, pages 199–201, 1942.

64 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations, The IBM Research Symposia Series*, pages 85–103. Springer, 1972. doi:10.1007/978-1-4684-2001-2_9.

65 Kim-Manuel Klein, Adam Polak, and Lars Rohwedder. On minimizing tardy processing time, max-min skewed convolution, and triangular structured ILPs. In *SODA 2023*, pages 2947–2960, 2023. doi:10.1137/1.9781611977554.CH112.

66 Dušan Knop, Martin Koutecký, Asaf Levin, Matthias Mnich, and Shmuel Onn. High-multiplicity N-fold IP via configuration LP. *Mathematical Programming*, 200(1):199–227, June 2023. doi:10.1007/s10107-022-01882-9.

67 Dušan Knop and Martin Koutecký. Scheduling meets n-fold integer programming. *J. Scheduling*, 21(5):493–503, 2018. doi:10.1007/S10951-017-0550-0.

68 Dušan Knop and Martin Koutecký. Scheduling kernels via configuration LP. In *ESA 2022*, volume 244 of *LIPIcs*, pages 73:1–73:15, 2022. doi:10.4230/LIPICS.ESA.2022.73.

69 Dušan Knop, Martin Koutecký, Asaf Levin, Matthias Mnich, and Shmuel Onn. Multitype integer monoid optimization and applications. *CoRR*, abs/1909.07326, 2019. arXiv:1909.07326.

70 Dušan Knop, Martin Koutecký, and Matthias Mnich. A unifying framework for manipulation problems. In *AAMAS 2018*, pages 256–264. International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA / ACM, 2018. URL: http://dl.acm.org/citation.cfm?id=3237427.

71 Dušan Knop, Martin Koutecký, and Matthias Mnich. Combinatorial n-fold integer programming and applications. *Math. Program.*, 184(1):1–34, 2020. doi:10.1007/s10107-019-01402-2.

72 Dušan Knop, Martin Koutecký, and Matthias Mnich. Voting and bribing in single-exponential time. *ACM Trans. Economics and Comput.*, 8(3):12:1–12:28, 2020. doi:10.1145/3396855.

73 Martin Koutecký, Asaf Levin, and Shmuel Onn. A parameterized strongly polynomial algorithm for block structured integer programs. In *Proc. ICALP 2018*, volume 107 of *LIPIcs*, pages 85:1–85:14, 2018. doi:10.4230/LIPICS.ICALP.2018.85.

74 Martin Koutecký and Nimrod Talmon. Multi-party campaigning. In *AAAI 2021*, pages 5506–5513. AAAI Press, 2021. doi:10.1609/AAAI.V35I6.16693.

75 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.

76 Andrea Lodi, Silvano Martello, and Michele Monaci. Two-dimensional packing problems: A survey. *European Journal of Operational Research*, 141(2):241–252, 2002. doi:10.1016/S0377-2217(02)00123-6.

77 Diane Maclagan. Antichains of monomial ideals are finite. *Proceedings of the American Mathematical Society*, 129(6):1609–1615, 2001.

78 Hugues Marchand, Alexander Martin, Robert Weismantel, and Laurence A. Wolsey. Cutting planes in integer and mixed integer programming. *Discret. Appl. Math.*, 123(1-3):397–446, 2002. doi:10.1016/S0166-218X(01)00348-1.

79 Alexander Martin. *Integer programs with block structure*. PhD thesis, Zuse Institute Berlin, 1999.

80 Theodore Samuel Motzkin. The multi-index transportation problem. In *Bulletin of the American Mathematical Society*, volume 58, pages 494–494, 1952.

81 Martin Nägele, Richard Santiago, and Rico Zenklusen. Congruency-constrained TU problems beyond the bimodular case. In *SODA 2022*, pages 2743–2790. SIAM, 2022. doi:10.1137/1.9781611977073.108.

82 Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity - Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012. doi:10.1007/978-3-642-27875-4.

83 Ivo Nowak. *Relaxation and Decomposition Methods for Mixed Integer Nonlinear Programming, Second Edition*, volume 152. Birkhäuser Basel, 2005. doi:10.1007/3-7643-7374-1.

84 John E Olson. A combinatorial problem on finite abelian groups, I. *Journal of number theory*, 1(1):8–10, 1969.

85 Joseph Paat, Robert Weismantel, and Stefan Weltge. Distances between optimal solutions of mixed-integer programs. *Math. Program.*, 179(1):455–468, 2020. doi:10.1007/s10107-018-1323-z.

86 Christos H. Papadimitriou. On the complexity of integer programming. *J. ACM*, 28(4):765–768, 1981. doi:10.1145/322276.322287.

87 Marcin Pilipczuk, Michał Pilipczuk, and Sebastian Siebertz. Lecture notes from the course "Sparsity" given at the Faculty of Mathematics, Informatics, and Mechanics of the University of Warsaw, Winter semesters 2017/18 and 2019/20. Available online at `https://www.mimuw.edu.pl/~mp248287/sparsity2`.

88 András Prékopa. *Stochastic programming*, volume 324 of *Mathematics and its Applications*. Kluwer Academic Publishers Group, Dordrecht, 1995.

89 Harilaos N. Psaraftis. A dynamic programming approach for sequencing groups of identical jobs. *Oper. Res.*, 28(6):1347–1359, 1980. `doi:10.1287/OPRE.28.6.1347`.

90 Victor Reis and Thomas Rothvoss. The subspace flatness conjecture and faster integer programming. In *FOCS 2023*, pages 974–988. IEEE, 2023. `doi:10.1109/FOCS57990.2023.00060`.

91 Francisco Santos and Bernd Sturmfels. Higher lawrence configurations. *J. Comb. Theory A*, 103(1):151–164, 2003. `doi:10.1016/S0097-3165(03)00092-X`.

92 Guntram Scheithauer and Johannes Terno. The modified integer round-up property of the one-dimensional cutting stock problem. *European J. Oper. Res.*, 84(3):562–571, 1995.

93 Éva Tardos. A strongly polynomial algorithm to solve combinatorial linear programs. *Operations Research*, 34(2):250–256, 1986. `doi:10.1287/OPRE.34.2.250`.

94 D. Antony Tarvin, R. Kevin Wood, and Alexandra M. Newman. Benders decomposition: Solving binary master problems by enumeration. *Oper. Res. Lett.*, 44(1):80–85, 2016. `doi:10.1016/J.ORL.2015.11.009`.

95 Paolo Toth and Daniele Vigo, editors. *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001.

96 J. M. van den Akker, J. A. Hoogeveen, and Steef L. van de Velde. Parallel machine scheduling by column generation. *Oper. Res.*, 47(6):862–872, 1999. `doi:10.1287/OPRE.47.6.862`.

97 Menkes van den Briel, Thomas Vossen, and Subbarao Kambhampati. Reviving integer programming approaches for AI planning: A branch-and-cut framework. In *Automated Planning and Scheduling, ICAPS 2005. Proceedings*, pages 310–319. AAAI, 2005. URL: `http://www.aaai.org/Library/ICAPS/2005/icaps05-032.php`.

98 François Vanderbeck and Martin W. P. Savelsbergh. A generic view of Dantzig-Wolfe decomposition in mixed integer programming. *Oper. Res. Lett.*, 34(3):296–306, 2006. `doi:10.1016/J.ORL.2005.05.009`.

99 Laurence A Wolsey. *Integer programming*. John Wiley & Sons, 2020.

Directed Disjoint Paths Remains W[1]-Hard on Acyclic Digraphs Without Large Grid Minors

Ken-ichi Kawarabayashi
National Institute of Informatics, Tokyo, Japan

Nicola Lorenz
Universität Hamburg, Germany

Marcelo Garlet Milani
National Institute of Informatics, Tokyo, Japan

Jacob Stegemann
Universität Hamburg, Germany

Abstract

In the Vertex-Disjoint-Paths-With-Congestion problem, the input consists of a digraph D, an integer c and k pairs of vertices (s_i, t_i), and the task is to find a set of paths connecting each s_i to its corresponding t_i, whereas each vertex of D appears in at most c many paths. The case where $c = 1$ is known to be NP-complete even if $k = 2$ [Fortune, Hopcroft and Wyllie, 1980] on general digraphs and is W[1]-hard with respect to k (excluding the possibility of an $f(k)n^{O(1)}$-time algorithm under standard assumptions) on acyclic digraphs [Slivkins, 2010]. The proof of [Slivkins, 2010] can also be adapted to show W[1]-hardness with respect to k for every congestion $c \geq 1$.

We strengthen the existing hardness result by showing that the problem remains W[1]-hard for every congestion $c \geq 1$ even if: **(1)** the input digraph D is acyclic, **(2)** D does not contain an acyclic $(5, 5)$-grid as a butterfly minor, **(3)** D does not contain an acyclic tournament on 9 vertices as a butterfly minor, and **(4)** D has ear-anonymity at most 5.

Further, we also show that the edge-congestion variant of the problem remains W[1]-hard for every congestion $c \geq 1$ even if: **(1)** the input digraph D is acyclic, **(2)** D has maximum undirected degree 3, **(3)** D does not contain an acyclic $(7, 7)$-wall as a weak immersion and **(4)** D has ear-anonymity at most 5.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Mathematics of computing → Paths and connectivity problems

Keywords and phrases digraphs, parameterized complexity, disjoint paths, butterfly minors, immersions, ear anonymity

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.2

Related Version *Full Version*: https://doi.org/10.48550/arXiv.2507.09868 [20]

Funding Research of Ken-ichi Kawarabayashi and Marcelo Garlet Milani supported by JSPS KAKENHI JP20A402 and 22H05001, and by JST ASPIRE JPMJAP2302.

1 Introduction

The Vertex-Disjoint-Paths (VDP) problem is a classic problem for directed graphs (digraphs) where the input consists of a digraph D and k vertex pairs (called terminal pairs) of the form (s_i, t_i), and the task is to decide whether k pairwise disjoint paths $P_1, P_2, \ldots, P_k$ exist in D such that each P_i is a path from s_i to t_i.

VDP is NP-complete even if $k = 2$ [15]. On directed acyclic graphs (DAGs), the problem can be solved in $n^{O(k)}$ time [15] and is W[1]-hard with respect to k (that is, there is no $f(k)n^{O(1)}$-time algorithm under standard assumptions) [27]. Under the assumption of the Exponential Time Hypothesis (ETH), no $n^{o(k)}$-time algorithm exists for VDP on DAGs [8], implying that the $n^{O(k)}$-time algorithm is essentially optimal.

© Ken-ichi Kawarabayashi, Nicola Lorenz, Marcelo Garlet Milani, and Jacob Stegemann;
licensed under Creative Commons License CC-BY 4.0

We investigate the impact on VDP on DAGs of excluding minors and imposing further structural restrictions on the input digraph. There are several notions of minors for digraphs, including topological minors, butterfly minors and strong minors. Strong minors are not meaningful on acyclic digraphs, and if a digraph D contains H as a topological minor, then it also contains H as a butterfly minor. Hence, out of the three relations above, butterfly minors are the most adequate for obtaining our results in the most general form.

Directed treewidth was introduced by [18], who also proved that VDP can be solved in $n^{f(k,\mathrm{dtw}(D))}$ time, where $\mathrm{dtw}(D)$ is the *directed treewidth* of the input digraph D. As the directed treewidth of DAGs is 0, this roughly generalizes the $n^{O(k)}$-time algorithm for DAGs mentioned above.

The edge-disjoint variant of the problem is called Edge-Disjoint-Paths (EDP). [3] proved that EDP can be solved in $f(k)n^{O(1)}$-time on Eulerian digraphs. EDP is W[1]-hard with respect to k even if the input DAG is planar [8]. We observe that the reduction from the edge-disjoint to the vertex-disjoint variant of EDP mentioned above does not preserve planarity, and so this hardness result does not immediately transfer to the vertex-disjoint variant. In fact, vertex-disjoint EDP is solvable in $f(k)n^{O(1)}$-time on planar digraphs [12].

Relaxations of VDP were considered in order to find tractable settings. One such relaxation is Vertex-Disjoint-Paths-With-Congestion (VDPwC), where each vertex/edge can be used by at most g many paths in the solution instead of requiring disjointness. The value g is referred to as the maximum *congestion* at a vertex/edge.

The hardness reduction due to [27] can be adapted to VDPwC, also yielding W[1]-hardness with respect to k on DAGs [1]. This was further strengthened by [1], who showed that, assuming the *Exponential Time Hypothesis* (ETH), for every computable function f and every constant $g \geq 1$, there is no $f(k)n^{o(k/\log k)}$-time algorithm for VDPwC on DAGs.

[28] studied an approximation variant of VDP where the goal is to either find a solution for the given instance or decide that no solution joining at least k/q terminal pairs exist, for some constant q. They proved that such a variant is W[1]-hard on DAGs for all constants q.

A maximization variant of VDP was also studied. Approximation properties of this variant have been considered by [6, 28, 4, 7, 5, 9, 22, 10, 14, 21].

The undirected analog of VDP is computationally much easier and can be solved in $f(k)n^3$ time for some function f [26]. The exponent of n was later improved to 2 [19] and recently to $1 + o(1)$ [23].

In rough terms, the algorithms of [26, 19, 23] above work by applying the *irrelevant vertex technique* in order to repeatedly remove vertices from the graph until the *treewidth* is sufficiently small so that the problem can be solved efficiently. Towards this end, the algorithms rely on the *Flat Wall Theorem* [26], which roughly states that, for every graph G, we have three cases: G contains a large clique as a minor, G contains a large *flat wall*, or G has bounded treewidth.

On a somewhat different direction, [25] introduced the parameter *ear anonymity*, a parameter which tries to capture the "structure" of maximal paths in a digraph, and asked if VDP can be solved in $f(k)n^{h(\mathrm{ea}(D))}$ time on DAGs, where $\mathrm{ea}(D)$ is the *ear anonymity* of the input digraph D.

We prove that VDPwC and its edge-disjoint variant remain computationally hard even if we impose structural restrictions as described above. We obtain additional lower bounds when assuming the Exponential Time Hypothesis (ETH). Below, $\overrightarrow{TT}_9$ is the unique acyclic orientation of the undirected clique on 9 vertices, and the acyclic $(5,5)$-grid is the orientation of the undirected $(5,5)$-grid where each edge is oriented from "left" to "right" and from "top" to "bottom".

▶ **Theorem 13.** *For every congestion $g \geq 1$, Vertex-Disjoint-Paths-With-Congestion is W[1]-hard with respect to the number k of terminal pairs, even if*

- *The input digraph D is acyclic,*
- *D contains no $\overrightarrow{TT}_9$ as a butterfly minor,*
- *D contains no acyclic $(5,5)$-grid as butterfly minor, and*
- $\mathrm{ea}(D) \leq 5$.

Furthermore, assuming the ETH, no $f(k)n^{o(\sqrt{k/g})}$ time algorithm exists for VDPwC, even under the above conditions.

Our reduction may create large complete bipartite digraphs where all edges are oriented from one partition to the other. Hence, it would be interesting to determine whether VDP remains W[1]-hard even on DAGs without large acyclic grids and without such complete bipartite digraphs as butterfly minors.

We also consider excluding *weak immersions* instead of butterfly minors, as immersions are more closely related to edge-disjoint than to vertex-disjoint paths. Since the maximum degree is closed under immersions, restricting the maximum degree of a digraph D also exclude the possibility of D having any digraph with larger degree as a weak immersion. For this reason, we consider here the acyclic wall instead of the acyclic grid, as the grid contains vertices of degree 4. We present the result above for the edge-disjoint variant of the problem, but we note that both reductions can be adapted to both variants of the problem.

▶ **Theorem 18** ($\star$). *For every congestion $g \geq 1$, Edge-Disjoint-Paths-With-Congestion is W[1]-hard with respect to the number k of terminal pairs, even if*

- *The input digraph D is acyclic,*
- *D has maximum undirected degree 3,*
- *D contains no acyclic $(7,7)$-wall as a weak immersion, and*
- $\mathrm{ea}(D) \leq 5$.

Furthermore, assuming the ETH, no $f(k)n^{o(\sqrt{k/g})}$ time algorithm exists for Edge-Disjoint-Paths-With-Congestion, even under the above conditions.

Observe that the hardness reductions of [27, 1] do not exclude the existence of a $\overrightarrow{TT}_k$ in the reduced instance, and the reduction due to [8] for the planar case does not exclude the existence of acyclic grids of size n.

The paper is organized as follows. Section 2 contains definitions and notation. In Section 3 we prove Theorem 13. Section 3.1 contains our hardness reduction, Sections 3.2–3.4 contain the structural analysis and Section 3.5 completes the proof. In Section 4 we prove Theorem 18. Due to space constraints, proofs marked with ($\star$) are deferred to full version of this manuscript.

2 Preliminaries

Sets. For $n \in \mathbb{N}$ we define $[n] \coloneqq \{1, \ldots, n\}$. A *multiset over a set X* is a function $f\colon X \to \mathbb{N}$. For every $x \in X$, the value $f(x)$ is the *multiplicity* of that element in the multiset. We write multisets as $\{x_1\colon f(x_1), x_2\colon f(x_2), \ldots, x_n\colon f(x_n)\}$, since this uniquely defines f. When explicitly stated to be a multiset, we may also write multisets like $\{x_1, x_2, \ldots, x_n\}$. In this case, the underlying set X is $\{x_1, x_2, \ldots, x_n\}$ (understood as a usual set) and the multiplicity of an element $x \in X$ is given by $f(x) \coloneqq |\{i \in [n] \mid x_i = x\}|$.

Digraphs. We refer the reader to [2] for an introduction to digraph theory. A directed graph (digraph) D is a tuple (V, E) where $E \subseteq V \times V$ and E does not contain *loops*, that is, edges of the form (v, v). We write $V(D) \coloneqq V$ and $E(D) \coloneqq E$.

Given a set $X \subseteq V(D)$, we write $D - X$ for the digraph $(Y \coloneqq V(D) \setminus X, E(D) \cap (Y \times Y))$. Similarly, given a set $F \subseteq E(D)$, we write $D - F$ for the digraph $(V(D), E(D) \setminus F)$.

If D is a digraph and $v \in V(D)$, then $\mathsf{in}_D(v) \coloneqq \{u \in V \mid (u, v) \in E(D)\}$ is the set of *in-neighbours* and $\mathsf{out}_D(v) \coloneqq \{u \in V \mid (v, u) \in E(D)\}$ the set of *out-neighbours* of v. By $\mathsf{indeg}_D(v) \coloneqq |\mathsf{in}(v)|$ we denote the *in-degree* of v and by $\mathsf{outdeg}_D(v) \coloneqq |\mathsf{out}(v)|$ its out-degree.

Given a vertex v in a digraph D, the *split* of v is the operation which consists of replacing v with two vertices $v_{\text{in}}, v_{\text{out}}$, adding the edge $(v_{\text{in}}, v_{\text{out}})$ and the edge $(u, v_{\text{in}}), (v_{\text{out}}, w)$ for each $u \in \mathsf{in}(v)$ and each $w \in \mathsf{out}(v)$.

Paths and walks. A *walk* of length ℓ in a digraph D is a sequence of vertices $W \coloneqq (v_0, v_1, \ldots, v_\ell)$ such that $(v_i, v_{i+1}) \subseteq E(D)$, for all $0 \leq i < \ell$. We say that W is a v_0-v_ℓ-walk.

A walk $W \coloneqq (v_0, v_1, \ldots, v_\ell)$ is called a *path* if no vertex appears twice in it and it is called a *cycle* if $v_0 = v_\ell$ and $v_i \neq v_j$ for all $0 \leq i < j < \ell$. We often identify a walk W in D with the corresponding subgraph and write $V(W)$ and $E(W)$ for the set of vertices and arcs appearing on it.

Given two walks $W_1 \coloneqq (x_1, x_2, \ldots, x_j)$ and $W_2 \coloneqq (y_1, y_2, \ldots, y_k)$, we write $W_1 \cdot W_2$ for the *concatenation* of W_1 and W_2, defined as follows. If $x_j = y_1$, $W_1 \cdot W_2 \coloneqq (x_1, x_2, \ldots, x_j, y_2, y_3, \ldots, y_k)$. If W_1 or W_2 is an empty sequence, then $W_1 \cdot W_2$ is the other walk (or the empty sequence if both walks are empty). Finally, if $x_j \neq y_1$, $W_1 \cdot W_2 \coloneqq W_1 \cdot (x_j, y_1) \cdot W_2$.

Given a walk $W \coloneqq (x_1, x_2, \ldots, x_n)$, we write Wx_i for the walk $(x_1, \ldots, x_i)$ and similarly $x_i W$ for the walk $(x_i, \ldots, x_n)$.

A digraph D without any cycles is called a *directed acyclic graph* (DAG). A vertex v in D is called a *source* if $\mathsf{indeg}_D(v) = 0$ and a *sink* if $\mathsf{outdeg}_D(v) = 0$.

Let $x_1, x_2, \ldots, x_n$ be an ordering of the vertices of a digraph D. We say that this ordering is a *topological ordering* of D if for every edge $(x_i, x_j) \in E(D)$ we have $i < j$. We note that a digraph admits a topological ordering if, and only if, it is a DAG.

Digraph classes. An *in-tree (out-tree) with root* $r \in V(D)$ is a DAG where for all $v \in V(D)$ there exists exactly one directed path from v to r (from r to v for out-trees).

The *transitive tournament on k vertices* is the digraph $\overrightarrow{TT}_k$ with vertex set $v_1, v_2, \ldots, v_k$ and edge set $\{(v_i, v_j) \mid 1 \leq i < j \leq k\}$.

We consider two types of directed grids (see Figure 1). An *acyclic (p, q)-grid* is the digraph with vertex set $\{(r, c) \mid 1 \leq r \leq p \text{ and } 1 \leq c \leq q\}$ and edge set $\{((r, c), (r + 1, c)) \mid 1 \leq r \leq p - 1 \text{ and } 1 \leq c \leq q\} \cup \{((r, c), (r, c + 1)) \mid 1 \leq r \leq p \text{ and } 1 \leq c \leq q - 1\}$.

An *acyclic (p, q)-wall* is the digraph obtained from an acyclic (p, q)-grid by applying the split operation to every vertex of indegree and outdegree exactly 2.

Immersions and minors. Let D and H be two digraphs. A weak immersion of H in D is a function μ with domain $V(H) \cup E(H)$ satisfying the following:

1. μ maps $V(H)$ injectively into $V(D)$,
2. $\mu((u, v))$ is a directed path from $\mu(u)$ to $\mu(v)$ in D for every $(u, v) \in E(H)$ and
3. the directed paths $\mu(e)$ and $\mu(f)$ are edge disjoint for every pair of distinct edges $e, f \in E(H)$.

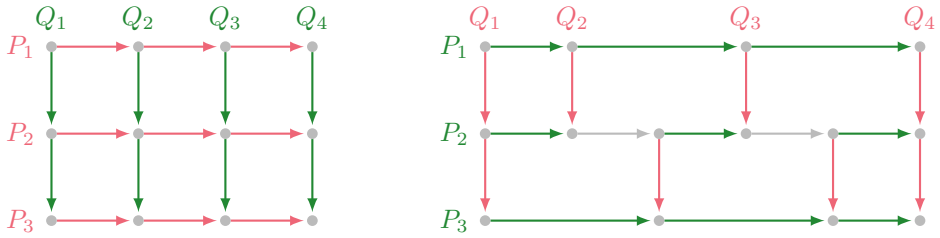


Figure 1 An acyclic $(3, 4)$-grid on the left and an acyclic $(3, 4)$-wall on the right.

Let H and D be directed graphs. A **butterfly-model** of H in D is a function μ which assigns to every $x \in V(H) \cup E(H)$ a subdigraph of D such that:

1. for every pair of distinct vertices $u, v \in V(H)$, $\mu(u)$ and $\mu(v)$ are vertex-disjoint,
2. for a vertex $v \in V(H)$ and a non-incident edge $e \in E(H)$, $\mu(v)$ and $\mu(e)$ are vertex-disjoint,
3. for every $v \in V(H)$, $\mu(v)$ is the union of an in-tree and an out-tree intersecting exactly on their common root, and
4. for every $(u, v) \in E(H)$, $\mu((u, v))$ is a directed path starting at a vertex of the out-tree of $\mu(u)$ and ending at a vertex of the in-tree of $\mu(v)$.

Additionally we define a *center* function $c_\mu \colon V(H) \to V(D)$ as follows. Let $v \in V(H)$. As $\mu(v)$ is an acyclic graph, it therefore admits a topological ordering. We let $c_\mu(v) \in V(\mu(v))$ be minimal in this ordering with the property that it can be reached by all sources of $\mu(v)$ (note that this does not depend on the choice of the topological ordering).

Note that $\mu(v)$ being a union of an in-tree and an out-tree intersecting exactly on their common root implies that $c_\mu(v)$ can also reach all sinks of $\mu(v)$.

Ear anonymity. An *ear* in a digraph D is a subgraph of D which is either a path or a cycle. Let P be an ear. A sequence $(a_1, a_2, \ldots, a_k)$ of arcs of P is an *identifying sequence for* P if $k \geq 1$ and every ear Q containing $(a_1, a_2, \ldots, a_k)$ in this order is a subgraph of P.

Let $P = (a_1, a_2, \ldots, a_k)$ be a maximal ear in a digraph D, given by its arc-sequence (in the case of a cycle, any arc of P can be chosen as a_1). The *ear anonymity* of P in D, denoted by $\mathrm{ea}_D(P)$, is the length of the shortest identifying sequence for P. If $k = 0$, we say that $\mathrm{ea}_D(P) = 0$. The ear anonymity of a digraph D, denoted by $\mathrm{ea}(D)$, is the maximum ear anonymity of the maximal ears of D.

Complexity theory. We refer the reader to [11, 13] for formal definitions of parameterized complexity concepts.

We skip a formal definition of $\mathsf{W}[1]$-*hardness* and recall only the properties that we require for our results. If a parameterized problem L_1 is $\mathsf{W}[1]$-hard with respect to k_1 and there is a reduction from L_1 to L_2 parameterized by k_2 with running time at most $f(k_1)n^{O(1)}$ such that $k_2 = h(k_1)$, then L_2 is $\mathsf{W}[1]$-hard with respect to k_2. Under standard assumptions, if a problem is $\mathsf{W}[1]$-hard with respect to some parameter k, then no algorithm with running time $f(k)n^{O(1)}$ exists for said problem.

The *Exponential Time Hypothesis* (ETH) states that 3-SAT on n variables and m clauses cannot be solved in $2^{o(n)} \cdot (n + m)^{O(1)}$ time [16, 17].

Computational problems. We recall the definitions of the following decision problems.

VERTEX-DISJOINT-PATHS-WITH-CONGESTION (VDPWC)

Input	A digraph D, two integers g and k and a multiset $T = \{(s_1,t_1),(s_2,t_2),\ldots,(s_k,t_k)\}$ of vertex pairs.
Question	Is there a multiset $\mathcal{L}$ of paths in D such that for each $(s_i,t_i) \in S$ there are $\text{mult}_T((s_i,t_i))$ many s_i-t_i paths in $\mathcal{L}$ and for each $v \in V(D)$ there are at most g paths in $\mathcal{L}$ which contain v?

EDGE-DISJOINT-PATHS-WITH-CONGESTION (EDPWC)

Input	A digraph D, two integers g and k and a multiset $T = \{(s_1,t_1),(s_2,t_2),\ldots,(s_k,t_k)\}$ of vertex pairs.
Question	Is there a multiset $\mathcal{L}$ of paths in D such that for each $(s_i,t_i) \in T$ there are $\text{mult}_T((s_i,t_i))$ many s_i-t_i paths in $\mathcal{L}$ and for each $e \in E(D)$ there are at most g paths in $\mathcal{L}$ which contain e?

The special cases where the congestion g above is 1 are called VERTEX-DISJOINT-PATHS and EDGE-DISJOINT-PATHS. Our main reduction is from the following W[1]-hard problem.

GRIDTILING

Input	Numbers $n,k \in \mathbb{N}$ and sets $S_{i,j} \subseteq [n] \times [n]$ for every $1 \le i,j \le k$.
Question	Are there $a_1,a_2,\ldots,a_k,b_1,b_2,\ldots,b_k \in [n]$ such that $(a_i,b_j) \in S_{i,j}$ holds for all $1 \le i,j \le k$?

▶ **Theorem 1** ([24, Lemma 1]). *GRIDTILING is W[1]-hard with respect to k. Furthermore, unless the ETH fails, there is no $f(k)n^{o(k)}$ time algorithm for GRIDTILING.*

3 Vertex disjoint paths

We provide a reduction from GRIDTILING to VDPWC. Our gadgets are inspired by the reduction due to [27].

3.1 Hardness reduction

▶ **Construction 2.** Let $(n,\{S_{i,j}\}_{i,j=1}^{k})$ be an instance of GRIDTILING. Let $g \ge 1$ be an integer. We construct a VERTEX-DISJOINT-PATHS-WITH-CONGESTION instance (D,T,g) as follows.

We use two types of *gadgets* in our construction: selector and verifier gadgets. For each row and each column, we have one selector gadget $\text{SEL}^{r,i}$ and $\text{SEL}^{c,i}$, respectively. For each $\circ \in \{r,c\}$ and each $i \in [k]$, we construct $\text{SEL}^{\circ,i}$ as follows (see Figure 2).

For each $j \in [k]$, add the source $s^{\circ,i,j}$ if $\circ = r$ and $s^{\circ,j,i}$ if $\circ = c$ instead.

Define two paths $P^{\circ,i}$ and $Q^{\circ,i}$ by their subpaths as follows. For each $1 \le \ell \le n$, define

$$P_\ell^{\circ,i} = p_{\ell,0}^{\circ,i},\ldots,p_{\ell,k}^{\circ,i} \qquad \text{and} \qquad Q_\ell^{\circ,i} = q_{\ell,1}^{\circ,i},\ldots,q_{\ell,k+1}^{\circ,i}.$$

Then,

$$P^{\circ,i} = P_1^{\circ,i} \cdot \ldots \cdot P_n^{\circ,i} \qquad \text{and} \qquad Q^{\circ,i} = Q_1^{\circ,i} \cdot \ldots \cdot Q_n^{\circ,i}.$$

Connect the paths above as follows. For each $\ell \in [n]$ and each $j \in [k]$, add the edges $(p_{\ell,0}^{\circ,i}, q_{\ell,k+1}^{\circ,i})$ and $(p_{\ell,j}^{\circ,i}, q_{\ell,j}^{\circ,i})$.

Add the terminal pairs $(p_{1,0}^{\circ,i}, p_{n,k}^{\circ,i})$ and $(q_{1,1}^{\circ,i}, q_{n,k+1}^{\circ,i})$ with multiplicity $g-1$ to T. Add the terminal pair $(p_{1,0}^{\circ,i}, q_{n,k+1}^{\circ,i})$ with multiplicity 1 to T.

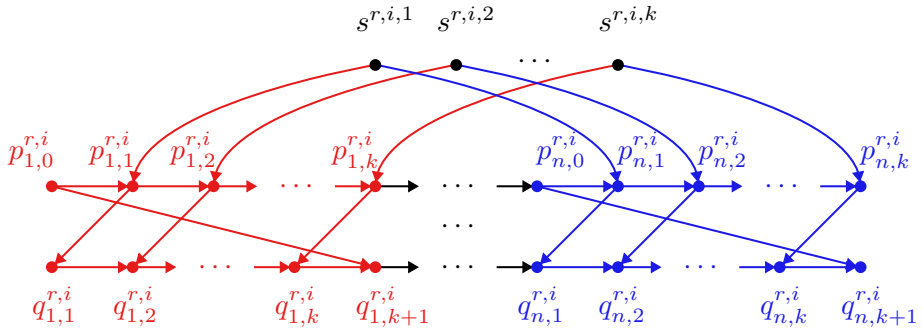


Figure 2 The selector gadget $\mathrm{SEL}^{r,i}$. The colored subgraphs are repeating building blocks.

This concludes the construction of the selector gadget. Vertices and edges of the selector can be summarized as follows.

$$V(\mathrm{SEL}^{\circ,i}) = V(P^{\circ,i}) \cup V(Q^{\circ,i}) \cup \left\{ s^{\circ,i,j} \;\middle|\; j \in [k] \right\},$$
$$E(\mathrm{SEL}^{\circ,i}) = E(P^{\circ,i}) \cup E(Q^{\circ,i})$$
$$\cup \left\{ (s^{\circ,i,j}, p^{\circ,i}_{\ell,j}), (p^{\circ,i}_{\ell,j}, q^{\circ,i}_{\ell,j}) \;\middle|\; j \in [k], \ell \in [n] \right\}$$
$$\cup \left\{ (p^{\circ,i}_{\ell,0}, q^{\circ,i}_{\ell,k+1}) \;\middle|\; \ell \in [n] \right\}.$$

Next, we construct the verifier gadgets $\mathrm{VER}^{i,j}$ for $1 \le i, j \le k$ (see Figure 3).

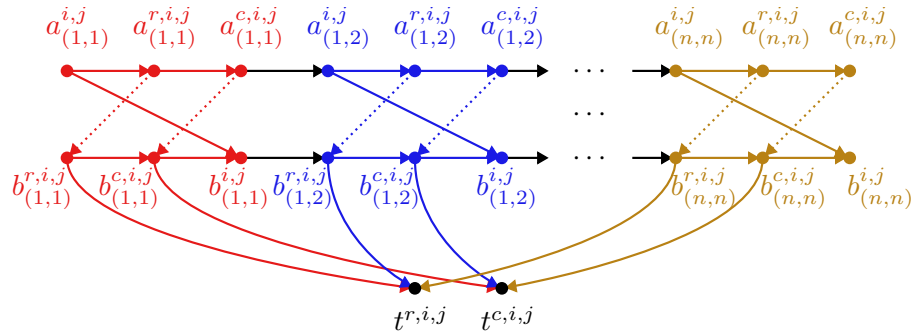


Figure 3 The verifier gadget $\mathrm{VER}^{i,j}$. The colored subgraphs are repeating building blocks, where the existence of the dashed arrows depends on the set $S_{i,j}$.

First, add the targets $t^{r,i,j}, t^{c,i,j}$.

Define two paths $A^{i,j}$ and $B^{i,j}$ by their subpaths as follows. For $d \in [n] \times [n]$, let

$$A^{i,j}_d = a^{i,j}_d \cdot a^{r,i,j}_d \cdot a^{c,i,j}_d \qquad \text{and} \qquad B^{i,j}_d = b^{r,i,j}_d \cdot b^{c,i,j}_d \cdot b^{i,j}_d.$$

For $1 \le x \le n$, let

$$A^{i,j}_x = A^{i,j}_{(x,1)} \cdot \ldots \cdot A^{i,j}_{(x,n)} \qquad \text{and} \qquad B^{i,j}_x = B^{i,j}_{(x,1)} \cdot \ldots \cdot B^{i,j}_{(x,n)}.$$

Now define

$$A^{i,j} = A^{i,j}_1 \cdot \ldots \cdot A^{i,j}_n \qquad \text{and} \qquad B^{i,j} = B^{i,j}_1 \cdot \ldots \cdot B^{i,j}_n.$$

Connect $A^{i,j}$ to $B^{i,j}$ and $B^{i,j}$ to $t^{r,i,j}, t^{c,i,j}$ by adding, for each $d \in S_{i,j}$, the edges $(a_d^{i,j}, b_d^{i,j})$, $(a_d^{r,i,j}, b_d^{r,i,j})$, $(a_d^{c,i,j}, b_d^{c,i,j})$, $(b_d^{r,i,j}, t^{r,i,j})$ and $(b_d^{c,i,j}, t^{c,i,j})$.

Add the terminal pairs $(a^{i,j}_{(1,1)}, a^{c,i,j}_{(n,n)})$ and $(b^{r,i,j}_{(1,1)}, b^{i,j}_{(n,n)})$ with multiplicity $g-1$ to T. Add the terminal pair $(a^{i,j}_{(1,1)}, b^{i,j}_{(n,n)})$ with multiplicity 1 to T.

This concludes the construction of the verifier gadgets. We can summarize the vertices and edges of the verifier as follows.

$$V(\mathrm{VER}^{i,j}) = V(A^{i,j}) \cup V(B^{i,j}) \cup \{t^{r,i,j}, t^{c,i,j}\}.$$

$$E(\mathrm{VER}^{i,j}) = E(A^{i,j}) \cup E(B^{i,j}) \cup \left\{ \begin{array}{lll} (a_d^{i,j}, b_d^{i,j}), & (a_d^{r,i,j}, b_d^{r,i,j}), & (a_d^{c,i,j}, b_d^{c,i,j}), \\ & (b_d^{r,i,j}, t^{r,i,j}), & (b_d^{c,i,j}, t^{c,i,j}) \end{array} \;\middle|\; d \in S_{i,j} \right\}.$$

Finally, connect the selectors and verifiers via additional edges as follows.

$$E' = \left\{ (q^{r,i}_{\ell,j}, a^{r,i,j}_{(\ell,x)}), (q^{c,j}_{\ell,i}, a^{c,i,j}_{(x,\ell)}) \;\middle|\; i,j \in [k] \text{ and } \ell, x \in [n] \right\}$$

This completes the construction of D. The multiset of terminal pairs is given by

$$\begin{aligned} T = &\left\{ (s^{r,i,j}, t^{r,i,j}) : 1, (s^{c,i,j}, t^{c,i,j}) : 1, (a^{i,j}_{(1,1)}, b^{i,j}_{n,n}) : 1 \;\middle|\; i,j \in [k] \right\} \\ &\cup \left\{ (p^{r,i}_{1,0}, q^{r,i}_{n,k+1}) : 1, (p^{c,i}_{1,0}, q^{c,i}_{n,k+1}) : 1 \;\middle|\; i \in [k] \right\} \\ &\cup \left\{ \begin{array}{ll} (p^{r,i}_{1,0}, p^{r,i}_{n,k}) : g-1, & (q^{r,i}_{1,1}, q^{r,i}_{n,k+1}) : g-1 \\ (p^{c,i}_{1,0}, p^{c,i}_{n,k}) : g-1, & (q^{c,i}_{1,1}, q^{c,i}_{n,k+1}) : g-1 \end{array} \;\middle|\; i \in [k] \right\} \\ &\cup \left\{ (a^{i,j}_{(1,1)}, a^{c,i,j}_{(n,n)}) : g-1, (b^{r,i,j}_{(1,1)}, b^{i,j}_{(n,n)}) : g-1 \;\middle|\; i,j \in [k] \right\} \end{aligned}$$

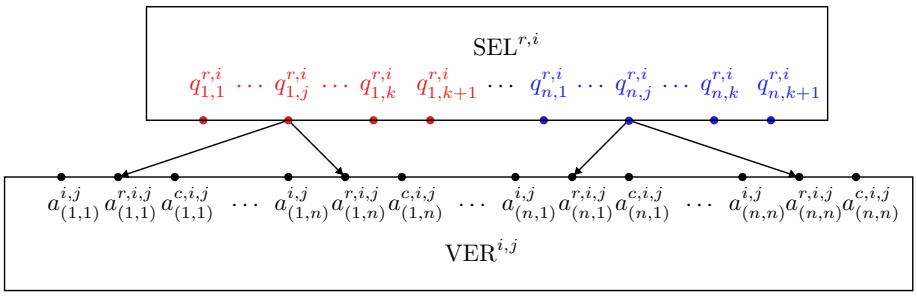


Figure 4 The edges between $\mathrm{SEL}^{r,i}$ and $\mathrm{VER}^{i,j}$. The colors in the selector again mark the repeating building blocks.

In total, D contains k selectors $\mathrm{SEL}^{r,i}$ and k selectors $\mathrm{SEL}^{c,i}$. Each selector contains $k + 2g - 1$ sources. There are k^2 verifier gadgets, each containing $2g - 1$ sources. Hence, we have in total $2k(k + 2g - 1) + k^2(2g - 1) = 2g(k^2 + k) + k^2 - 2k$ terminal pairs in the constructed instance.

Before analyzing the structural properties of the digraph constructed above, we show that our reduction is correct.

▶ **Lemma 3.** *Let $I_G := (n, \{S_{i,j}\}_{i,j=1}^k)$ be an instance of* GridTiling. *Let $g \geq 1$ be an integer. Let $I_L := (D, T, g)$ be the* VDPwC *instance obtained from Construction 2. Then I_G is a* yes *instance if, and only if, I_L is a* yes *instance.*

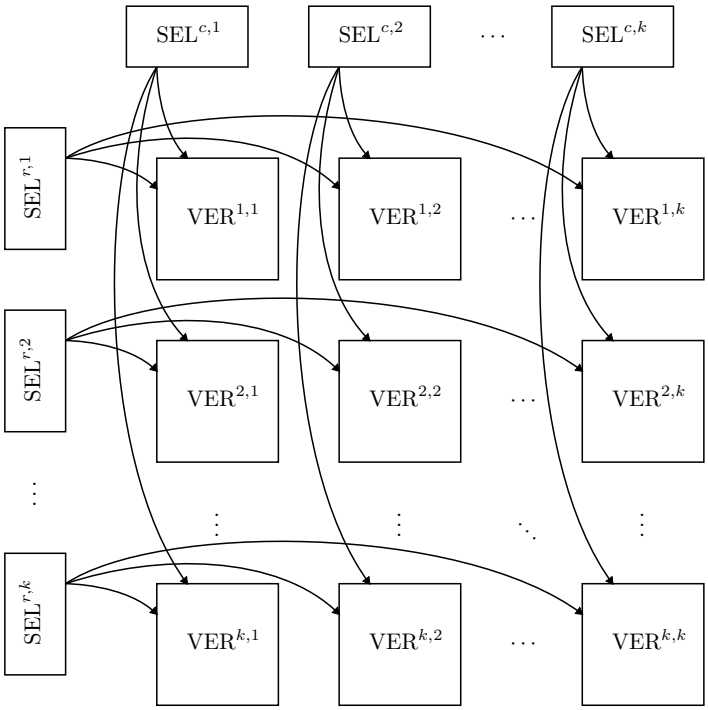


Figure 5 A simplified representation of all edges between selectors and verifiers. Each edge represents in fact multiple edges as shown in Figure 4.

Proof.

First direction. If I_G is a *yes* instance, then I_L is a *yes* instance.

Let $a_1, a_2, \dots a_k, b_1, b_2, \dots, b_k \in [n]$ be a solution for I_G. We construct a solution for I_L as follows. Recall from Construction 2 that the set of terminal pairs is given by

$$\begin{aligned} T^* = &\left\{ \begin{array}{ll} (p^{r,i}_{1,0}, p^{r,i}_{n,k}) : g-1, & (q^{r,i}_{1,1}, q^{r,i}_{n,k+1}) : g-1 \\ (p^{c,i}_{1,0}, p^{c,i}_{n,k}) : g-1, & (q^{c,i}_{1,1}, q^{c,i}_{n,k+1}) : g-1 \end{array} \;\middle|\; i \in [k] \right\} \\ &\cup \left\{ (a^{i,j}_{(1,1)}, a^{c,i,j}_{(n,n)}) : g-1, (b^{r,i,j}_{(1,1)}, b^{i,j}_{(n,n)}) : g-1 \;\middle|\; i,j \in [k] \right\}, \\ T = &\left\{ (s^{r,i,j}, t^{r,i,j}) : 1, (s^{c,i,j}, t^{c,i,j}) : 1, (a^{i,j}_{(1,1)}, b^{i,j}_{n,n}) : 1 \;\middle|\; i,j \in [k] \right\} \\ &\cup \left\{ (p^{r,i}_{1,0}, q^{r,i}_{n,k+1}) : 1, (p^{c,i}_{1,0}, q^{c,i}_{n,k+1}) : 1 \;\middle|\; i \in [k] \right\} \cup T^*. \end{aligned}$$

For each $i, j \in [k]$ and each $h \in [g-1]$, we define paths connecting the terminal pairs.

First, we consider the terminal pairs whose corresponding path does not depend on the solution for I_G.

For each $\circ \in \{r, c\}$, the pair $(p^{\circ,i}_{(0,1)}, p^{\circ,i}_{(n,k)})$ gets connected via $g-1$ copies of the path $(p^{\circ,i}_{(1,0)}, p^{\circ,i}_{(1,1)}) \cdot P^{\circ,i} \cdot (p^{\circ,i}_{(n,k-1)}, p^{\circ,i}_{(n,k)})$ and the pair $(q^{\circ,i}_{(1,1)}, q^{\circ,i}_{(n,k+1)})$ gets connected via $g-1$ copies of the path $(q^{\circ,i}_{(1,1)}, q^{\circ,i}_{(1,2)}) \cdot Q^{\circ,i} \cdot (q^{\circ,i}_{(n,k)}, q^{\circ,i}_{(n,k+1)})$.

The pair $(a^{i,j}_{(1,1)}, a^{i,j,c}_{(n,n)})$ gets connected via $g-1$ copies of the path $(a^{i,j}_{(1,1)}, a^{r,i,j}_{(1,1)}) \cdot A^{i,j} \cdot (a^{r,i,j}_{(n,n)}, a^{c,i,j}_{(n,n)})$ and the pair $(b^{i,j}_{(1,1)}, b^{i,j}_{(n,n)})$ gets connected via $g-1$ copies of the path $(b^{r,i,j}_{(1,1)}, b^{c,i,j}_{(1,1)}) \cdot B^{i,j} \cdot (b^{c,i,j}_{(n,n)}, b^{i,j}_{(n,n)})$.

Let $\mathcal{L}^*$ be the linkage connecting the terminal pairs above. Observe that $\mathcal{L}^*$ provokes a congestion of $g-1$ on every vertex in $P^{r,i}, P^{c,i}, Q^{r,i}, Q^{c,i}, A^{i,j}$ and $B^{i,j}$.

The solution for the remaining pairs will depend on the values a_i and b_j above. Recall that the remaining terminal pairs all have multiplicity 1 in T.

The terminal pair $(s^{r,i,j}, t^{r,i,j})$ gets connected via the path

$$L^{r,i,j} \coloneqq (s^{r,i,j}, p^{r,i}_{a_i,j}, q^{r,i}_{a_i,j}, a^{r,i,j}_{(a_i,b_j)}, b^{r,i,j}_{(a_i,b_j)}, t^{r,i,j}).$$

The terminal pair $(s^{c,i,j}, t^{c,i,j})$ gets connected via the path

$$L^{c,i,j} \coloneqq (s^{c,i,j}, p^{c,j}_{b_j,i}, q^{c,j}_{b_j,i}, a^{c,i,j}_{(a_i,b_j)}, b^{c,i,j}_{(a_i,b_j)}, t^{c,i,j}).$$

Observe that, because $(a_i, b_j) \in S_{i,j}$, we know that the edges $(a^{r,i,j}_{(a_i,b_j)}, b^{r,i,j}_{(a_i,b_j)})$ and $(a^{c,i,j}_{(a_i,b_j)}, b^{c,i,j}_{(a_i,b_j)})$ do in fact exist.

The terminal pair $(a^{i,j}, b^{i,j})$ gets connected via the path

$$L^{i,j} \coloneqq (a^{i,j}_{(1,1)}, a^{r,i,j}_{(1,1)}) \cdot A^{i,j} \cdot (a^{i,j}_{(a_i,b_j)}, b^{i,j}_{(a_i,b_j)}) \cdot B^{i,j} \cdot (b^{c,i,j}_{(n,n)}, b^{i,j}_{(n,n)}).$$

The terminal pair $(p^{r,i}, q^{r,i})$ gets connected via the path

$$L^{r,i} \coloneqq (p^{r,i}, p^{r,i}_{1,0}) \cdot P^{r,i} \cdot (p^{r,i}_{a_i,0}, q^{r,i}_{a_i,k+1}) \cdot Q^{r,i} \cdot (q^{r,i}_{n,k+1}, q^{r,i}).$$

The terminal pair $(p^{c,j}, q^{c,j})$ gets connected via the path

$$L^{c,j} \coloneqq (p^{c,j}, p^{c,j}_{1,0}) \cdot P^{c,j} \cdot (p^{c,j}_{b_j,0}, q^{c,j}_{b_j,k+1}) \cdot Q^{c,j} \cdot (q^{c,j}_{n,k+1}, q^{c,j}).$$

These paths are pairwise vertex disjoint. Further, each path visits at most one vertex in each $P^{r,i}, P^{c,i}, Q^{r,i}, Q^{c,i}, A^{i,j}$ and $B^{i,j}$. Hence, together with the linkage $\mathcal{L}^*$, we obtain congestion at most g in order to connect all terminal pairs of T.

Second direction. If I_L is a *yes* instance, then I_G is a *yes* instance.

Let $\mathcal{L}$ be a solution for I_L. For each $i, j \in [k]$, let $L^{r,i,j}$, $L^{c,i,j}$, $L^{r,i}$, $L^{c,j}$ and $L^{i,j}$ be the paths connecting the pairs $(s^{r,i,j}, t^{r,i,j})$, $(s^{c,i,j}, t^{c,i,j})$, $(p^{r,i}_{0,1}, q^{r,i}_{n,k+1})$, $(p^{c,j}_{0,1}, q^{c,j}_{n,k+1})$ and $(a^{i,j}_{(1,1)}, b^{i,j}_{(1,1)})$, respectively, and let $\mathcal{L}' \subseteq \mathcal{L}$ be the set containing these paths. Let $\mathcal{L}^* \subseteq \mathcal{L}$ be the multiset of paths connecting the pairs in T^*.

Observe that, for each $L \in \mathcal{L}^*$, the path L is the unique path in D from $\mathsf{start}(L)$ to $\mathsf{end}(L)$, as there are no paths from $Q^{r,i}$ to $P^{r,i}$ in the selector gadgets and also no path from $B^{i,j}$ to $A^{i,j}$ in the verifier gadgets. Hence, $\mathcal{L}^*$ provokes a congestion of $g-1$ on every vertex in $P^{r,i}, P^{c,i}, Q^{r,i}, Q^{c,i}, A^{i,j}$ and $B^{i,j}$.

As every non-terminal vertex inside $\mathcal{L}'$ lies in $P^{r,i}, P^{c,i}, Q^{r,i}, Q^{c,i}, A^{i,j}$ or $B^{i,j}$, we deduce that the paths in $\mathcal{L}'$ are pairwise vertex disjoint, as otherwise we would have congestion greater than g at some vertex.

Let $i \in [k]$ and consider the path $L^{r,i} \in \mathcal{L}'$ connecting $(p^{r,i}_{0,1}, q^{r,i}_{n,k+1})$. Note that the set of vertices which can reach $q^{r,i}_{n,k+1}$ and are reachable from $p^{r,i}_{0,1}$ is given by $\{p^{r,i}_{0,1}, q^{r,i}_{n,k+1}\} \cup V(P^{r,i}) \cup V(Q^{r,i})$. Hence, $L^{r,i}$ lies inside $\mathrm{SEL}^{r,i}$. Further, it must contain exactly one edge from $P^{r,i}$ to $Q^{r,i}$, as there are no edges from $Q^{r,i}$ back to $P^{r,i}$.

Let v_p be the last vertex along $L^{r,i}$ which lies on $P^{r,i}$ and let v_q be the first vertex along $L^{r,i}$ which lies on $Q^{r,i}$. Observe that there is exactly one path from $p^{r,i}_{0,1}$ to v_p, which we call P', and exactly one path from v_q to $q^{r,i}_{n,k+1}$, which we call Q'. Because of the linkage $\mathcal{L}^*$, the congestion at P' and at Q' is equal to g, and so no other path of $\mathcal{L}'$ can intersect P' or Q'.

Let $j \in [k]$ and let v_j be the first vertex of $P^{r,i}$ along $L^{r,i,j}$. By construction, v_j is necessarily of the form $p^{r,i}_{a_j,j}$, for some $a_j \in [n]$. As $L^{r,i,j}$ is disjoint from P', the last vertex of P' must occur before $p^{r,i}_{a_j,j}$ along $P^{r,i}$.

Similarly, let u_j be the first vertex of $Q^{r,i}$ along $L^{r,i,j}$. As $L^{r,i,j}$ is disjoint from Q', the first vertex of Q' must occur after u_j along $Q^{r,i}$. Additionally, u_j must occur at or after $q^{r,i}_{a_j,j}$ along $Q^{r,i}$, as earlier vertices are not reachable from v_j.

This implies that $L^{r,i}$ must contain the edge $(p^{r,i}_{a_j,0}, q^{r,i}_{a_j,k+1})$. As this holds for every $j \in [k]$ and every $L^{r,i,j}$, we have that $a_j = a_\ell$ for all $j, \ell \in [k]$. Let $a_i \in [n]$ be the value equal to all a_j. Also, for each $j \in [k]$, the path $L^{r,i,j}$ leaves the gadget $\mathrm{SEL}^{r,i}$ at the vertex $q^{r,i}_{a_i,j}$.

By an analogous argument, we obtain a value b_j from the selector $\mathrm{SEL}^{c,j}$ and conclude that, for every $i \in [k]$, the path $L^{c,i,j}$ leaves $\mathrm{SEL}^{c,j}$ at the vertex $q^{c,i,j}_{b_j,i}$. We claim that $a_1, a_2, \ldots, a_k, b_1, b_2, \ldots, b_k$ is a solution for I_G.

Let $i, j \in [k]$. We show that $(a_i, b_j) \in S_{i,j}$. As shown above, the path $L^{r,i,j}$ enters $\mathrm{VER}^{i,j}$ by an edge of the form $(q^{r,i}_{a_i,j}, a^{r,i,j}_{a_i,x})$, for some $x \in [n]$. Analogously, the path $L^{c,i,j}$ enters $\mathrm{VER}^{i,j}$ by an edge of the form $(q^{c,j}_{b_j,i}, a^{c,i,j}_{\ell,b_j})$, for some $\ell \in [n]$.

By an analogous argument as before, we conclude that the path $L^{i,j}$ must contain exactly one edge of the form $(a^{i,j}_{\ell',x'}, b^{i,j}_{\ell',x'})$, as otherwise it would intersect both $L^{r,i,j}$ and $L^{c,i,j}$, causing congestion greater than g at some vertex. In particular, this implies that $\ell' = a_i = \ell$ and $x' = b_j = x$. Hence, $L^{r,i,j}$ must contain the edge $(a^{r,i,j}_{a_i,b_j}, b^{r,i,j}_{a_i,b_j})$ and $L^{c,i,j}$ must contain the edge $(a^{c,i,j}_{a_i,b_j}, b^{c,i,j}_{a_i,b_j})$. This is only possible if $(a_i, b_j) \in S_{i,j}$, as otherwise these edges do not exist in D.

Hence, for all $i, j \in [k]$, we have $(a_i, b_j) \in S_{i,j}$, and so I_G is a *yes* instance. ◀

3.2 Excluding an acyclic $(5,5)$-grid

▶ **Lemma 4** ($\star$). *For each $1 \leq i, j \leq k$, none of the gadgets $\mathrm{VER}^{i,j}$, $\mathrm{SEL}^{r,i}$ and $\mathrm{SEL}^{c,i}$ contain any acyclic $(3,3)$-grid as a butterfly minor.*

The following observation can easily be verified from the construction and will be useful in subsequent proofs.

▶ **Observation 5.** *Let $(n, \{S_{i,j}\}_{i,j=1}^k)$ be a GridTiling instance. Let (D,T) be the instance given by Construction 2. For all $i, i', j, j' \in [k]$ and all $\circ \in \{r, c\}$, the following holds*

1. *there are no edges leaving $V(\mathrm{VER}^{i,j})$,*
2. *there are no edges coming into $V(\mathrm{SEL}^{\circ,i})$,*
3. *if there is an edge from $V(\mathrm{SEL}^{r,i'})$ to $V(\mathrm{VER}^{i,j})$, then $i' = i$, and*
4. *if there is an edge from $V(\mathrm{SEL}^{c,j'})$ to $V(\mathrm{VER}^{i,j})$, then $j' = j$.*

▶ **Corollary 6.** *Let $(n, \{S_{i,j}\}_{i,j=1}^k)$ be an instance of GridTiling. Let (D,T) be the reduced instance constructed by Construction 2. Let H be an acyclic digraph with exactly one source and exactly one sink. If H is a butterfly minor of D with model μ, then there exist $i, j \in [n]$ such that $\mathrm{Im}(\mu) \subseteq D[V(\mathrm{SEL}^{r,i}) \cup V(\mathrm{VER}^{i,j})]$ or $\mathrm{Im}(\mu) \subseteq D[V(\mathrm{SEL}^{c,j}) \cup V(\mathrm{VER}^{i,j})]$.*

We can now exclude an acyclic $(5,5)$-grid by arguing that such a grid in the constructed digraph would imply the existence of an $(3,3)$-grid in one of the selectors or one of the verifiers.

▶ **Lemma 7.** *Let $(n, \{S_{i,j}\}_{i,j=1}^k)$ be an instance of GridTiling. Let (D,T) be the reduced instance constructed by Construction 2. Then D does not contain any acyclic $(5,5)$-grid as a butterfly minor.*

Proof. We assume towards a contradiction that D contains an acyclic $(5,5)$-grid G as a butterfly minor. Let μ be the model of G in D.

G has exactly one source and one sink. By Corollary 6, there exist $i, j \in [n]$ such that $\mathrm{Im}(\mu) \subseteq D[V(\mathrm{SEL}^{r,i}) \cup V(\mathrm{VER}^{i,j})]$ or $\mathrm{Im}(\mu) \subseteq D[V(\mathrm{SEL}^{c,j}) \cup V(\mathrm{VER}^{i,j})]$. Without loss of generality we assume that $\mathrm{Im}(\mu) \subseteq D[V(\mathrm{SEL}^{r,i}) \cup V(\mathrm{VER}^{i,j})]$.

Define a partition of the vertices of G as follows

$$A \coloneqq \left\{v \in V(G) \;\middle|\; c_\mu(v) \in V(\mathrm{SEL}^{r,i})\right\},$$
$$B \coloneqq \left\{v \in V(G) \;\middle|\; c_\mu(v) \in V(\mathrm{VER}^{i,j})\right\}.$$

By Observation 5, there is no edge from B to A, as otherwise the model would give a path from $c_\mu(b) \in V(\mathrm{VER}^{i,j})$ to $c_\mu(a) \in V(\mathrm{SEL}^{r,i})$ for some $b \in B$ and $a \in A$.

Let v be the vertex in the center of the acyclic $(5,5)$-grid G.

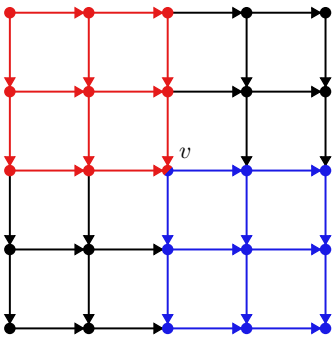


Figure 6 The acyclic $(5,5)$-grid G, $v \in V(G)$ and two acyclic $(3,3)$-subgrids (red and blue).

If $v \in A$, then the top left acyclic $(3,3)$-subgrid (see the red colored subgraph in Figure 6) is also in A as there is no edge from B to A. Since there is no edge from $\mathrm{VER}^{i,j}$ to $\mathrm{SEL}^{r,i}$, restricting the domain of μ to this acyclic $(3,3)$-grid and its codomain to $\mathrm{SEL}^{r,i}$ would give a model for the acyclic $(3,3)$-grid as a butterfly minor in $\mathrm{SEL}^{r,i}$. But by Lemma 4, $\mathrm{SEL}^{r,i}$ does not contain any acyclic $(3,3)$-grid as a butterfly minor. Similarly, if $v \in B$, then the bottom right acyclic $(3,3)$-subgrid (see the blue colored subgraph in Figure 6) is also in B. We would get an acyclic $(3,3)$-grid as a butterfly minor in $\mathrm{VER}^{i,j}$. But by Lemma 4, $\mathrm{VER}^{i,j}$ does not contain any acyclic $(3,3)$-grid as a butterfly minor. Hence, we obtain a contradiction. ◀

3.3 Excluding a $\overrightarrow{TT}_9$

▶ **Lemma 8.** *For each $1 \leq i, j \leq k$, none of the gadgets $\mathrm{VER}^{i,j}$, $\mathrm{SEL}^{r,i}$ and $\mathrm{SEL}^{c,i}$ contain any $\overrightarrow{TT}_6$ as a butterfly minor.*

Proof. We prove the statement for the selector $\mathrm{SEL}^{r,i}$. The proof for the remaining gadgets follow analogously. We assume towards a contradiction that $\mathrm{SEL}^{r,i}$ contains a $G \coloneqq \overrightarrow{TT}_6$ as a butterfly minor. Let μ be the model of G in $\mathrm{SEL}^{r,i}$.

Let s be the vertex of G with in-degree 0 and $U \coloneqq V(G - s)$. Then, the indegree of all vertices $u \in U$ is at least 1. Therefore, $c_\mu(u) \neq s^{r,i,j}$ for all $1 \leq j \leq k$.

Define $A \coloneqq V(P^{r,i}), B \coloneqq V(Q^{r,i})$. The five vertices of U lie in $A \cup B$. So $|U \cap A| \geq 3$ or $|U \cap B| \geq 3$.

Without loss of generality, $\{u, w, v\} \in U \cap A$ and $c_\mu(u), c_\mu(w)$ and $c_\mu(v)$ appear in this order on $P^{r,i}$. Then every path from $c_\mu(u)$ to $c_\mu(v)$ contains $c_\mu(w)$. Thus, $\mu(w)$ is not disjoint from $\mu(u) \cup \mu((u, v)) \cup \mu(v)$. A contradiction. The case where $|U \cap B| \geq 3$ follows analogously. ◀

▶ **Lemma 9** (⋆). *Let $(n, \{S_{i,j}\}_{i,j=1}^k)$ be an instance of* GridTiling. *Let (D, T) be the reduced instance constructed by Construction 2. Then D contains no $\overrightarrow{TT}_9$ as a butterfly minor.*

The analysis of Lemma 9 is essentially tight, as shown by the observation below.

▶ **Observation 10.** *Let $n \geq 5$ and k be a natural number and define $S_{i,j} \coloneqq [n] \times [n]$ for all $i, j \in [k]$ as an instance of* GridTiling. *Let (D, T) be the reduced instance constructed by Construction 2. Then D contains a $\overrightarrow{TT}_8$ as a butterfly minor as shown in Figure 7.*

3.4 Bounded ear anonymity

▶ **Lemma 11.** *Let $(n, \{S_{i,j}\}_{i,j=1}^k)$ be an instance of* GridTiling. *Let (D, T) be the reduced instance constructed by Construction 2. Then* $\mathrm{ea}(D) \leq 5$.

Proof. Let P be a maximal path in D. Let s be the starting point and t the endpoint of P. We construct an ear identify sequence $\bar{a}$ for P as follows.

Let $\mathcal{H}$ be the set of paths of the form $P^{\circ,i}, Q^{\circ,i}, A^{i,j}$ or $B^{i,j}$ in D. Let F be the set of edges leaving some path in $\mathcal{H}$ and entering a different path in $\mathcal{H}$. The next claim follows easily from the construction of D.

▷ Claim 12. Let $(v, u) \in F$ be an edge from some path R_1 to another path R_2, where R_1 is some path of the form $P^{\circ,i}$, $Q^{\circ,i}$ or $A^{i,j}$, and R_2 is some path of the form $Q^{\circ,i}, A^{i,j}$ or $B^{i,j}$. Let $v_1 \in V(R_1)$ be a vertex which can reach v and let $u_2 \in V(R_2)$ be a vertex which can be reached by u. Then there is exactly one v_1-u_2 path in D using (v, u).

Let e_s be the outgoing edge of s in P and let e_t be the incoming edge of t in P. Add e_s to $\bar{a}$. Add every arc in $F \cap E(P)$ to $\bar{a}$. Add e_t to $\bar{a}$. As $|F \cap E(P)| \leq 3$, we have that $|\bar{a}| \leq 5$.

We show that P is the unique maximal path of D visiting $\bar{a}$ in order. Observe that every edge leaving some source in D must end in some path in $\mathcal{H}$. Similarly, every edge entering some sink in D must start at some path in $\mathcal{H}$. Let $R_1 \in \mathcal{H}$ be the path in which e_s ends and let $R_2 \in \mathcal{H}$ be the path in which e_t starts.

If $R_1 = R_2$, then there is exactly one path in D from the head of e_s to the tail of e_t. Otherwise, there is an edge in $F \cap E(P)$. Let P' be the shortest subpath of P containing all edges in $F \cap E(P)$. By Claim 12, P' is the unique minimal path in D visiting those edges Further, by construction there is a unique path from the head of e_s to the start of P' and a unique path from the end of P' to the tail of e_t. Hence, P is the unique path in D visiting the edges of $\bar{a}$ in order. As $|\bar{a}| \leq 5$, we conclude that $\mathrm{ea}(D) \leq 5$. ◀

3.5 W[1]-hardness

▶ **Theorem 13.** *For every congestion $g \geq 1$,* Vertex-Disjoint-Paths-With-Congestion *is W[1]-hard with respect to the number k of terminal pairs, even if*

- *The input digraph D is acyclic,*
- *D contains no $\overrightarrow{TT}_9$ as a butterfly minor,*
- *D contains no acyclic $(5, 5)$-grid as butterfly minor, and*
- $\mathrm{ea}(D) \leq 5$.

Furthermore, assuming the ETH, no $f(k)n^{o(\sqrt{k/g})}$ time algorithm exists for VDPwC, even under the above conditions.

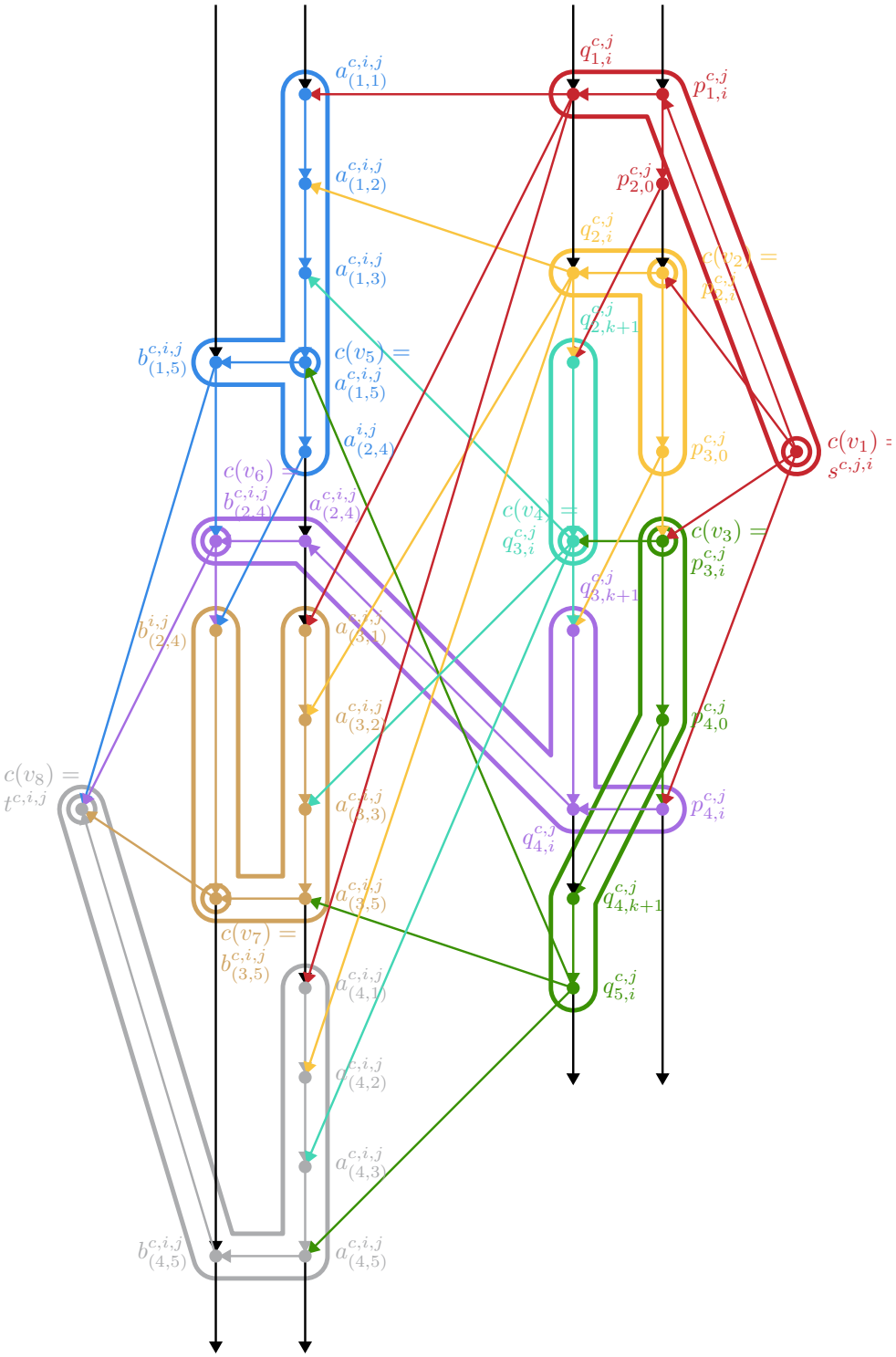


Figure 7 A butterfly model μ of the $\overrightarrow{TT}_8$ in D. Subdivided edges are still drawn as straight lines. The left part is $\mathrm{VER}^{i,j}$ and the right part is $\mathrm{SEL}^{c,j}$. The subgraph $\mu(v_i)$ is indicated by the colored region containing the center $c_\mu(v_i)$ and the path $\mu((v_i, v_j))$ is indicated by the paths leaving and entering the regions and is colored as $\mu(v_i)$.

Proof. We provide a reduction from GridTiling, which is W[1]-hard with respect to k by Theorem 1. Let $I_G = (n, \{S_{i,j}\}_{i,j=1}^{k})$ be a GridTiling instance and let $g \geq 1$ be an integer. We construct a VDPwC instance $I_L = (D, T, g)$ by using Construction 2. By Lemma 3, I_L is a *yes* instance if, and only if, I_G is a *yes* instance. By Lemmas 7 and 9, D contains no $\overrightarrow{TT}_9$ and no acyclic $(5,5)$-grid as a butterfly minor. Further, D is acyclic. Additionally, by Lemma 11, $\mathrm{ea}(D) \leq 5$. Finally, $|T| \in O(gk^2)$, and so the existence of an $f(|T|)n^{o(\sqrt{|T|/g})}$-time algorithm for VDPwC would imply the existence of an $f(k)n^{o(k)}$ algorithm for GridTiling, contradicting Theorem 1. ◀

We observe that VDPwC can be solved in polynomial time if $\mathrm{ea}(D) = 1$, as in this case every s_i-t_i path is disjoint from every s_j-t_j path if $i \neq j$, and so taking the shortest s_i-t_i path for every terminal pair (s_i, t_i) yields a solution if any exist.

Further, $\mathrm{ea}(\overrightarrow{TT}_k) = \lceil \frac{k-2}{2} \rceil$ and the acyclic (p,p)-grid has ear anonymity $p-1$. Since ear anonymity is closed under taking butterfly minors on DAGs [25], bounding $\mathrm{ea}(D)$ also excludes the presence of large acyclic grids and $\overrightarrow{TT}_k$ as butterfly minors.

4 Edge disjoint paths

4.1 Hardness reduction

▶ **Construction 14.** Let (D', T', g) be an instance of VDPwC. We construct an EDPwC instance (D, T, g) iteratively as follows. Apply the split operation to every vertex $v \in V(D')$, obtaining two vertices v_{in} and v_{out} in D.

If $\mathsf{indeg}_{D'}(v) \geq 3$, add a minimal in-tree T_v^{in} with $\mathsf{indeg}_{D'}(v)$ leaves where each vertex of the in-tree has indegree at most 2. Identify each vertex in $\mathsf{in}_D(v_{\text{in}})$ with a distinct leaf of T_v^{in} and remove the edges from $\mathsf{in}_D(v_{\text{in}})$ to v_{in}. Finally, add an edge from the root of T_v^{in} to v_{in}.

If $\mathsf{outdeg}_{D'}(v) \geq 3$, we proceed in an analogous fashion as above, adding an out-tree T_v^{out} instead. This completes the construction of D. Note that $\mathsf{outdeg}_D(v) + \mathsf{indeg}_D(v) \leq 3$ holds for all $v \in V(D)$.

The multiset of terminal pairs T is obtained from T' by replacing every occurrence of $(s,t) \in T'$ with $(s_{\text{in}}, t_{\text{out}})$.

4.2 Excluding a (7, 7)-wall

It is not too difficult to see that the digraph D given by Construction 14 contains an acyclic $(5,5)$-wall as an immersion. To exclude the acyclic $(7,7)$-wall, we use the fact that the original instance D' does not contain an acyclic $(5,5)$-grid.

▶ **Lemma 15** (⋆). *Let (D', T', g) be a VDPwC instance. Let (D, T, g) be the EDPwC instance given by Construction 14. If D contains an acyclic $(7,7)$-wall as a weak immersion, then D' contains an acyclic $(5,5)$-grid as a butterfly minor.*

4.3 Bounded ear anonymity

▶ **Observation 16** (⋆). *Let (D', T', g) be a VDPwC instance where D' is a DAG. Let (D, T, g) be the EDPwC instance given by Construction 14. Let P be a maximal path in D and let $P_1 \cdot P_2 \cdot \ldots \cdot P_n = P$ be a partition of P into subpaths such that P_i is a maximal subpath of P contained inside $T_{v_i}^{in}, (v_{i,in}, v_{i,out})$ and $T_{v_i}^{out}$ for some $v_i \in V(D')$. Let P' be the path obtained by replacing every P_i in P by v_i. Then P' is a path in D'.*

▶ **Lemma 17** (⋆). *Let (D', T', g) be a VDPwC instance. Let (D, T, g) be the EDPwC instance given by Construction 14. Then* $\mathrm{ea}(D) \leq \mathrm{ea}(D')$.

4.4 W[1]-hardness

The last remaining step for our hardness result is to show that the reduction given by Construction 14 is correct.

▶ **Theorem 18** (⋆). *For every congestion $g \geq 1$, Edge-Disjoint-Paths-With-Congestion is W[1]-hard with respect to the number k of terminal pairs, even if*

- *The input digraph D is acyclic,*
- *D has maximum undirected degree 3,*
- *D contains no acyclic $(7,7)$-wall as a weak immersion, and*
- $\mathrm{ea}(D) \leq 5$.

Furthermore, assuming the ETH, no $f(k)n^{o(\sqrt{k/g})}$ time algorithm exists for Edge-Disjoint-Paths-With-Congestion, even under the above conditions.

References

1 Saeed Akhoondian Amiri, Stephan Kreutzer, Dániel Marx, and Roman Rabinovich. Routing with congestion in acyclic digraphs. *Inf. Process. Lett.*, 151, 2019. doi:10.1016/J.IPL.2019.105836.

2 Jørgen Bang-Jensen and Gregory Z. Gutin. *Digraphs - Theory, Algorithms and Applications, Second Edition*. Springer Monographs in Mathematics. Springer, 2009.

3 Dario Giuliano Cavallaro, Ken-ichi Kawarabayashi, and Stephan Kreutzer. Edge-disjoint paths in eulerian digraphs. 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 704–715. ACM, 2024. doi:10.1145/3618260.3649758.

4 Parinya Chalermsook, Bundit Laekhanukit, and Danupon Nanongkai. Pre-reduction graph products: Hardnesses of properly learning dfas and approximating EDP on dags. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 444–453. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.54.

5 Chandra Chekuri, Sanjeev Khanna, and F. Bruce Shepherd. Edge-disjoint paths in planar graphs. In *45th Symposium on Foundations of Computer Science (FOCS 2004), 17-19 October 2004, Rome, Italy, Proceedings*, pages 71–80. IEEE Computer Society, 2004. doi:10.1109/FOCS.2004.27.

6 Chandra Chekuri, Sanjeev Khanna, and F. Bruce Shepherd. An O(sqrt(n)) approximation and integrality gap for disjoint paths and unsplittable flow. *Theory Comput.*, 2(7):137–146, 2006. doi:10.4086/TOC.2006.V002A007.

7 Chandra Chekuri, Sanjeev Khanna, and F. Bruce Shepherd. Edge-disjoint paths in planar graphs with constant congestion. *SIAM J. Comput.*, 39(1):281–301, 2009. doi:10.1137/060674442.

8 Rajesh Chitnis. A tight lower bound for edge-disjoint paths on planar dags. *SIAM J. Discret. Math.*, 37(2):556–572, 2023. doi:10.1137/21M1395089.

9 Julia Chuzhoy, David H. K. Kim, and Shi Li. Improved approximation for node-disjoint paths in planar graphs. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 556–569. ACM, 2016. doi:10.1145/2897518.2897538.

10 Julia Chuzhoy, David H. K. Kim, and Rachit Nimavat. New hardness results for routing on disjoint paths. *SIAM J. Comput.*, 51(2):17–189, 2022. doi:10.1137/17M1146580.

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 Marek Cygan, Dániel Marx, Marcin Pilipczuk, and Michal Pilipczuk. The planar directed k-vertex-disjoint paths problem is fixed-parameter tractable. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 197–206. IEEE Computer Society, 2013. doi:10.1109/FOCS.2013.29.

13 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.

14 Alina Ene, Matthias Mnich, Marcin Pilipczuk, and Andrej Risteski. On routing disjoint paths in bounded treewidth graphs. In Rasmus Pagh, editor, *15th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2016, June 22-24, 2016, Reykjavik, Iceland*, volume 53 of *LIPIcs*, pages 15:1–15:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICS.SWAT.2016.15.

15 Steven Fortune, John Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10(2):111–121, 1980. doi:10.1016/0304-3975(80)90009-2.

16 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.

17 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.

18 Thor Johnson, Neil Robertson, Paul Seymour, and Robin Thomas. Directed tree-width. *Journal of Combinatorial Theory, Series B*, 82(1):138–154, 2001. doi:10.1006/JCTB.2000.2031.

19 Ken-ichi Kawarabayashi, Yusuke Kobayashi, and Bruce A. Reed. The disjoint paths problem in quadratic time. *J. Comb. Theory B*, 102(2):424–435, 2012. doi:10.1016/J.JCTB.2011.07.004.

20 Ken-ichi Kawarabayashi, Nicola Lorenz, Marcelo Garlet Milani, and Jacob Stegemann. Directed disjoint paths remains w[1]-hard on acyclic digraphs without large grid minors. *CoRR*, abs/2507.09868, 2025. doi:10.48550/arXiv.2507.09868.

21 Jon M. Kleinberg and Éva Tardos. Approximations for the disjoint paths problem in high-diameter planar networks. *J. Comput. Syst. Sci.*, 57(1):61–73, 1998. doi:10.1006/JCSS.1998.1579.

22 Stavros G. Kolliopoulos and Clifford Stein. Approximating disjoint-path problems using packing integer programs. *Math. Program.*, 99(1):63–87, 2004. doi:10.1007/S10107-002-0370-6.

23 Tuukka Korhonen, Michal Pilipczuk, and Giannos Stamoulis. Minor containment and disjoint paths in almost-linear time. In *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024, Chicago, IL, USA, October 27-30, 2024*, pages 53–61. IEEE, 2024. doi:10.1109/FOCS61266.2024.00014.

24 Dániel Marx. A tight lower bound for planar multiway cut with fixed number of terminals. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part I*, volume 7391 of *Lecture Notes in Computer Science*, pages 677–688. Springer, 2012. doi:10.1007/978-3-642-31594-7_57.

25 Marcelo Garlet Milani. Directed ear anonymity. In José A. Soto and Andreas Wiese, editors, *LATIN 2024: Theoretical Informatics - 16th Latin American Symposium, Puerto Varas, Chile, March 18-22, 2024, Proceedings, Part II*, volume 14579 of *Lecture Notes in Computer Science*, pages 77–97. Springer, 2024. doi:10.1007/978-3-031-55601-2_6.

26 Neil Robertson and Paul D. Seymour. Graph minors .xiii. the disjoint paths problem. *J. Comb. Theory B*, 63(1):65–110, 1995. doi:10.1006/JCTB.1995.1006.

27 Aleksandrs Slivkins. Parameterized tractability of edge-disjoint paths on directed acyclic graphs. *SIAM Journal on Discrete Mathematics*, 24(1):146–157, 2010. doi:10.1137/070697781.

28 Michal Wlodarczyk. Constant approximating disjoint paths on acyclic digraphs is w[1]-hard. 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 57:1–57:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ISAAC.2024.57.

Parameterized Maximum Node-Disjoint Paths

Michael Lampis
Université Paris-Dauphine, PSL University, CNRS UMR7243, LAMSADE, Paris, France

Manolis Vasilakis
Université Paris-Dauphine, PSL University, CNRS UMR7243, LAMSADE, Paris, France

Abstract

We revisit the Maximum Node-Disjoint Paths problem, the natural optimization version of the famous Node-Disjoint Paths problem, where we are given an undirected graph G, k (demand) pairs of vertices (s_i, t_i), and an integer ℓ, and are asked whether there exist at least ℓ vertex-disjoint paths in G whose endpoints are given pairs. This problem has been intensely studied from both the approximation and parameterized complexity point of view and is notably known to be intractable by standard structural parameters, such as tree-depth, as well as the combined parameter ℓ plus pathwidth. We present several results improving and clarifying this state of the art, with an emphasis towards FPT approximation.

Our main positive contribution is to show that the problem's intractability can be overcome using approximation: We show that for several of the structural parameters for which the problem is hard, most notably tree-depth, the problem admits an *efficient FPT approximation scheme*, returning a $(1-\varepsilon)$-approximate solution in time $f(\mathrm{td}, \varepsilon)n^{\mathcal{O}(1)}$. We manage to obtain these results by comprehensively mapping out the structural parameters for which the problem is FPT if ℓ is also a parameter, hence showing that understanding ℓ as a parameter is key to the problem's approximability. This, in turn, is a problem we are able to solve via a surprisingly simple color-coding algorithm, which relies on identifying an insightful problem-specific variant of the natural parameter, namely the number of vertices used in the solution.

The results above are quite encouraging, as they indicate that in some situations where the problem does not admit an FPT algorithm, it is still solvable almost to optimality in FPT time. A natural question is whether the FPT approximation algorithm we devised for tree-depth can be extended to pathwidth. We resolve this negatively, showing that under the *Parameterized Inapproximability Hypothesis* no FPT approximation scheme for this parameter is possible, even in time $f(\mathrm{pw}, \varepsilon)n^{g(\varepsilon)}$. We thus precisely determine the parameter border where the problem transitions from "hard but approximable" to "inapproximable".

Lastly, we strengthen existing lower bounds by replacing W[1]-hardness by XNLP-completeness for parameter pathwidth, and improving the $n^{o(\sqrt{\mathrm{td}})}$ ETH-based lower bound for tree-depth to (the optimal) $n^{o(\mathrm{td})}$.

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

Keywords and phrases ETH, Maximum Node-Disjoint Paths, Parameterized Complexity, PIH

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.3

Related Version *Full Version*: `https://arxiv.org/abs/2404.14849`

Funding This work is partially supported by ANR project ANR-21-CE48-0022 (S-EX-AP-PE-AL).

1 Introduction

One of the most important problems of structural graph theory has arguably been Node-Disjoint Paths, where given a graph G and k pairs of its vertices (s_i, t_i) for $i = 1, \ldots, k$, called demands, the goal is to determine whether there exist k vertex-disjoint paths connecting s_i and t_i. This extensively studied problem [1, 4, 5, 20, 22, 24, 25, 32, 34] is one of the very first to be proven NP-complete (for k being part of the input) [19], and it has a central

© Michael Lampis and Manolis Vasilakis;
licensed under Creative Commons License CC-BY 4.0

role in the field of structural graph theory as well as in parameterized complexity [27], as the breakthrough result by Robertson and Seymour [31] that it is fixed-parameter tractable (FPT) parameterized by k (i.e., admits an $f(k)n^{\mathcal{O}(1)}$ algorithm for some function f, where $n = |V(G)|$) is the culmination of their long and influential series of works on Graph Minors.

In this work we concern ourselves with Maximum Node-Disjoint Paths (MaxNDP), the natural generalization of Node-Disjoint Paths where one asks whether at least $\ell \leq k$ demands can be routed by vertex-disjoint paths (we say that a demand is routed when there exists a path connecting its endpoints in the set of vertex-disjoint paths of the solution). Notice that one could alternatively phrase this as an optimization problem and ask for the maximum number of demands that can be routed. Even though MaxNDP has been intensely studied with respect to its approximability [7, 8, 9, 10, 21], our understanding regarding its tractability under the perspective of parameterized complexity is rather limited. Given the rich literature regarding Node-Disjoint Paths and the importance of its structural parameterizations (indeed, Scheffler's $\mathrm{tw}^{\mathcal{O}(\mathrm{tw})}n^{\mathcal{O}(1)}$ algorithm [32] is a key ingredient of the proof of [31]), the quest to study MaxNDP under the same point of view is strongly motivated, with the hope of extending some of these results to it. Alas, prior work by Ene, Mnich, Pilipczuk, and Risteski [14] shows that MaxNDP is already W[1]-hard when parameterized by the tree-depth of the input graph (in fact their proof implies hardness for the combined parameter vertex integrity[1] plus feedback vertex number). On the other hand, notice that MaxNDP is trivially FPT by k; one can simply reduce it to 2^k instances of Node-Disjoint Paths. A natural question arising from this observation is whether a parameterization by ℓ renders the problem tractable. In this spirit, Marx and Wollan [28] studied this setting and proved that the problem is W[1]-hard even when parameterized by the combined parameter ℓ plus the treewidth of the input graph; a closer look into their proof reveals that their result extends to graphs of bounded pathwidth plus feedback vertex number. This plethora of negative results fails to answer which parameterizations render the problem tractable, and whether a parameterization by ℓ plus some structural parameter (larger than or incomparable to treewidth) may lift it to FPT.

Our Contribution. In the present paper we thoroughly investigate the complexity of MaxNDP under different parameterizations, and determine exactly when it is rendered tractable, by additionally employing the use of approximation in the process (see Figure 1 for a synopsis of our results). We start by showing that the problem is FPT parameterized by the number of vertices of an optimal solution by developing a simple algorithm that makes use of the color-coding technique introduced by Alon, Yuster, and Zwick [2]. We then prove that, albeit simple, this algorithm is in fact sufficient to pinpoint exactly when the problem is fixed-parameter tractable: utilizing a variety of structural observations, we develop FPT algorithms for various parameterizations (most involving ℓ as a parameter) at the core of all of which lies the previously mentioned algorithm. Along the way we also develop an FPT algorithm for the parameterization by the feedback edge number. These positive results, in conjunction with the hardness results of [14, 28], clearly showcase the transition of the problem from FPT to W[1]-hard for its various parameterizations (with the exception of cluster vertex deletion number, where it is unknown whether the problem is W[1]-hard).

Given the apparent hardness of the problem, we move on to consider it under the perspective of parameterized approximation (see [15] for a survey of the area). Here, we observe that utilizing the previously developed FPT algorithms when ℓ is also a parameter, can in fact lead to efficient FPT approximation schemes (FPT-ASes) in case of solely structural

[1] A graph has vertex integrity at most k if there exists a set of at most p vertices such that their deletion results in a graph with connected components of size at most $k - p$.

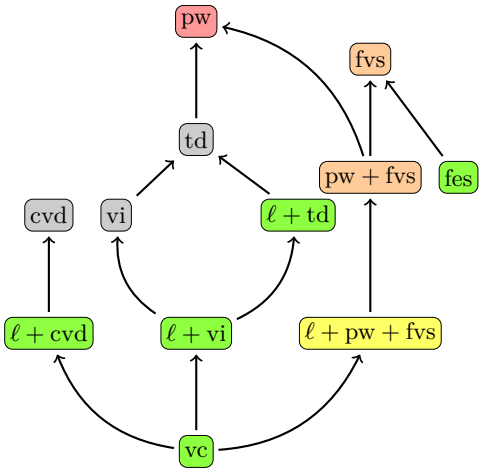


Figure 1 Our results and hierarchy of the related graph parameters, where vc, vi, td, pw, fvs, cvd, and fes stand for vertex cover, vertex integrity, tree-depth, pathwidth, feedback vertex number, cluster vertex deletion number, and feedback edge number respectively. For any graph, if the parameter at the tail of an arrow is a constant, that is also the case for the one at its head. Green and gray indicate that the problem is FPT (Theorems 2 and 3) and that it admits an FPT approximation scheme (Theorems 4 and 5) respectively. Yellow indicates W[1]-hardness, orange that there is no FPT approximation scheme (Theorem 6), and red XNLP-completeness (Theorem 8). Prior to this work, it was only known that the problem is W[1]-hard parameterized by vi + fvs [14] and by ℓ + pw + fvs [28].

parameterizations of the problem, when parameterized by the cluster vertex deletion number, the vertex integrity, or the tree-depth of the input graph; in the latter two cases the problem is known to be W[1]-hard [14].

The FPT approximation schemes developed indicate that the W[1]-hardness can, in some cases, be overcome via the use of approximation. Given the relationship of our studied parameters as well as the FPT-AS for tree-depth, a natural question is whether an analogous approximation scheme exists for the parameterization by pathwidth as well. Notice that the W[1]-hardness by ℓ plus the pathwidth pw and the feedback vertex number fvs of the input graph already excludes the existence of an approximation scheme of running time $f(\text{pw}, \text{fvs}, \varepsilon)n^{\mathcal{O}(1)}$, yet one of time $f(\text{pw}, \text{fvs}, \varepsilon)n^{g(\varepsilon)}$ remains possible. Our next result is to exclude the existence of such a scheme under the *Parameterized Inapproximability Hypothesis* [26], which was recently proved to hold under the ETH [17], and thus precisely determine the parameter border where the problem transitions from "hard but approximable" to "inapproximable". By slightly modifying our reduction, we subsequently show that the problem is XNLP-complete when parameterized solely by the pathwidth of the input graph, where XNLP is a complexity class that has been recently brought forth by Bodlaender, Groenland, Nederlof, and Swennenhuis [3], and such a result implies W[t]-hardness for all integers $t \geq 1$.

Lastly, we proceed to a more fine-grained examination of the hardness of MaxNDP when parameterized by the tree-depth of the input graph. Standard dynamic programming techniques can be used to obtain an $n^{\mathcal{O}(\text{tw})}$ algorithm, while previous work by Ene, Mnich, Pilipczuk, and Risteski [14] implies that the problem cannot be solved in time $n^{o(\sqrt{\text{td}})}$ under the ETH for graphs of tree-depth td, thereby leaving hope for an $n^{o(\text{td})}$ algorithm. We revisit said proof, and by employing a recursive structure introduced by Lampis and Vasilakis [23] we bridge this gap and prove that MaxNDP cannot be solved in time $n^{o(\text{td})}$ under the ETH, rendering the $n^{\mathcal{O}(\text{tw})}$ algorithm optimal even for this much smaller class of graphs.

Related Work. Even though MaxNDP has been well-studied under the scope of approximation algorithms, the 20 years old algorithm of ratio $\mathcal{O}(\sqrt{n})$ due to Kolliopoulos and Stein [21] remains the state of the art in general graphs. This has been improved in the case of grid [6] and planar graphs [7], resulting in approximation ratios $\tilde{\mathcal{O}}(n^{1/4})$ and $\tilde{\mathcal{O}}(n^{9/19})$ respectively, where standard $\tilde{\mathcal{O}}$ notation is used to hide polylogarithmic terms. For graphs of pathwidth pw, Ene, Mnich, Pilipczuk, and Risteski [14] have presented an algorithm of approximation ratio $\mathcal{O}(\text{pw}^3)$. Regarding inapproximalibity results, after a series of works, Chuzhoy, Kim, and Nimavat [9, 10] have shown that the problem cannot be approximated in polynomial time (i) within a factor of $2^{\mathcal{O}(\log^{1-\varepsilon} n)}$ for any constant ε, assuming NP $\not\subseteq$ DTIME($n^{\text{polylog}n}$), and (ii) within a factor of $n^{\mathcal{O}(1/(\log\log n)^2)}$, assuming that there exists some constant $\delta > 0$ such that NP $\not\subseteq$ DTIME(2^{n^δ}).

The problem has been also studied under a parameterized complexity perspective. As already noted, it is trivially FPT by k by a simple reduction to 2^k instances of Node-Disjoint Paths, which is well-known to be FPT by the number of demands. On the other hand, Marx and Wollan [28] have shown that it becomes W[1]-hard when parameterized by $\ell + \text{tw}$, with a closer look into their proof revealing that their result extends to the parameterization by $\ell + \text{pw} + \text{fvs}$. Regarding structural parameterizations, Ene, Mnich, Pilipczuk, and Risteski [14] have proved that the problem is W[1]-hard when parameterized by the tree-depth of the input graph; in fact, their proof extends to graphs of bounded vertex integrity plus feedback vertex number. Fleszar, Mnich, and Spoerhase [16] have proposed an algorithm of running time $(k + \text{fvs})^{\mathcal{O}(\text{fvs})} n^{\mathcal{O}(1)}$ as an alternative to the $2^k \text{fvs}^{\mathcal{O}(\text{fvs})} n^{\mathcal{O}(1)}$ algorithm obtained by reducing the instance to Node-Disjoint Paths and then using Scheffler's algorithm [32].

Organization. In Section 2 we discuss the general preliminaries. Subsequently, in Section 3 we present various tractability results, followed by the inapproximability result for pathwidth in Section 4. Moving on, in Sections 5 and 6 we present the XNLP-completeness and the refined W[1]-hardness of the problem, when parameterized by the pathwidth and the tree-depth of the input graph respectively. Lastly, in Section 7 we present the conclusion as well as some directions for future research. Proofs of statements marked with ($\star$) are omitted and can be found in the full version.

2 Preliminaries

For $x, y \in \mathbb{Z}$, let $[x, y] = \{z \in \mathbb{Z} : x \leq z \leq y\}$, while $[x] = [1, x]$. For a set S, let $\binom{S}{c}$ denote the set of subsets of S of size c for some $c \in \mathbb{N}$, that is $\binom{S}{c} = \{S' \subseteq S : |S'| = c\}$. Throughout the paper we use standard graph notations [12] and assume familiarity with the basic notions of parameterized complexity [11]. All graphs considered are undirected without loops unless explicitly mentioned otherwise. Let $G = (V, E)$ be a graph. The *cluster vertex deletion number* of G, denoted $\text{cvd}(G)$, is the size of the smallest vertex set whose removal results in a cluster graph, i.e., a union of cliques. The *vertex integrity* of G, denoted $\text{vi}(G)$, is the minimum integer k such that there is a vertex set $S \subseteq V$ such that $|S| + \max_{C \in \mathsf{cc}(G-S)} |V(C)| \leq k$, where $\mathsf{cc}(G - S)$ denotes the set of connected components in $G - S$.

Given a graph $G = (V, E)$ and a partition of V into k independent sets $V_1, \ldots, V_k$, each of size n, k-Multicolored Clique asks whether G contains a k-clique, and is well-known to be W[1]-hard and not to admit any $f(k)n^{o(k)}$ algorithm, where f is any computable function, unless the ETH is false [11]. Multicolored Densest k-Subgraph asks for the maximum

number of edges that are induced by a subgraph of G that contains exactly one vertex per independent set (color). Let $E^{i,j} \subseteq E$ denote the set of edges $e = \{u, v\}$ where $u \in V_i$ and $v \in V_j$. In that case, let $s_i = |\{j \in [k] : E^{i,j} \neq \varnothing\}|$ and $\sigma = \sum_{i \in [k]} s_i/2$. Assuming that OPT denotes said maximum value, the *Parameterized Inapproximability Hypothesis* (PIH) [26] states that there exists a constant $0 < c < 1$ such that no $f(k)n^{\mathcal{O}(1)}$ algorithm can distinguish between OPT $= \sigma$ and OPT $< c \cdot \sigma$. In fact, one can assume without loss of generality that $1 \leq s_i \leq 3$ for all $i = 1, \ldots, k$ [26, Lemma 4.4]. This hypothesis was very recently proved to hold under the ETH [17] (see also [18]), and is the analogue of the PCP theorem in the setting of parameterized complexity.

Lastly, we give a formal definition of the problem this work is concerned with.

MAXIMUM NODE-DISJOINT PATHS

Instance: Graph $G = (V, E)$, set of k demand pairs $\mathcal{M} \subseteq \binom{V}{2}$, and integer $\ell \leq k$.

Goal: Determine whether at least ℓ demand pairs can be routed, where to route a pair we need to select a path connecting it, so that all selected paths are vertex-disjoint.

Notice that in the above definition, even though demand pairs may indeed share a terminal, the paths comprising a feasible solution must be vertex-disjoint, and this constraint also applies to their endpoints. Given an instance $\mathcal{I} = (G, \mathcal{M}(, \ell))$ of the optimization (or decision) version of MAXNDP and G' a subgraph of G, OPT$(\mathcal{I}[G'])$ denotes the maximum number of demands of $\mathcal{M}$ that can be routed in G'. We write OPT$(\mathcal{I})$ as a shorthand for OPT$(\mathcal{I}[G])$.

3 FPT Algorithms and Approximation Schemes

Here we present various tractability results for MAXNDP. We start by proving in Section 3.1 that the problem is FPT when parameterized by the number of vertices involved in an optimal solution; using this as well as some structural observations, we obtain FPT algorithms when parameterized by vc, $\ell +$ cvd, $\ell +$ vi, and $\ell +$ td. We additionally develop an FPT algorithm for the parameterization by fes. Moving on, in Section 3.2 we obtain FPT approximation schemes for various structural parameterizations of the problem by making use of the previous FPT algorithms; for most of said parameterizations the problem is known to be W[1]-hard.

3.1 Exact Algorithms

We start with the following theorem.

▶ **Theorem 1.** *Let $\mathcal{I} = (G, \mathcal{M}, \ell)$ be an instance of* MAXNDP. *Additionally, let τ be such that there exists a family $\mathcal{P}$ of vertex-disjoint paths each routing a demand of $\mathcal{M}$, where $|\mathcal{P}| = \min\{\ell, \text{OPT}(\mathcal{I})\}$ and $\sum_{P \in \mathcal{P}} |V(P)| \leq \tau$, with $|V(P)|$ denoting the number of vertices in path P. There is an algorithm that, given $\mathcal{I}$ and τ, decides $\mathcal{I}$ in time $2^{\mathcal{O}(\tau)}n^{\mathcal{O}(1)}$.*

Proof. Let $C = [\tau]$ be a set of τ colors. Randomly color the vertices of G with colors from C, and let A be the event where every one of the at most τ vertices of $\mathcal{P}$ receives a distinct color. Then, it follows that

$$\Pr[A] \geq \frac{\tau!}{\tau^\tau} > \frac{\left(\frac{\tau}{e}\right)^\tau}{\tau^\tau} = e^{-\tau},$$

therefore event A holds with probability at least $e^{-\tau}$.

Now, let $T[S]$ be equal to the maximum number of demands that can be routed by paths using only vertices of colors belonging to $S \subseteq C$, where each color is used in at most one path. Moreover, let $f(S) = 1$ if there exists at least one demand that can be routed using only vertices of colors belonging to S and 0 otherwise. Notice that f can be computed in polynomial time. Then, it holds that

$$T[S] = \max_{S' \subseteq S} \{f(S') + T[S \setminus S']\},$$

thus one can compute $T[C]$ in time $2^{\mathcal{O}(\tau)}n^{\mathcal{O}(1)}$ for a given coloring of the vertices of G.

By repeating this procedure $2^{\mathcal{O}(\tau)}$ times, with high probability there exists some iteration where event A holds, and the total running time is $2^{\mathcal{O}(\tau)}n^{\mathcal{O}(1)}$. By using standard techniques [11, Section 5.6], one can derandomize the described algorithm and obtain a deterministic one of the same running time. ◀

Using Theorem 1 we can obtain various parameterized algorithms, by bounding the number of vertices of an optimal solution using some simple observations.

▶ **Theorem 2** (⋆). *Given an instance $\mathcal{I} = (G, \mathcal{M}, \ell)$ of* MaxNDP*, there exist algorithms that decide $\mathcal{I}$ in time*

- $2^{\mathcal{O}(\mathrm{vc})}n^{\mathcal{O}(1)}$,
- $2^{\mathcal{O}(\mathrm{cvd}+\ell)}n^{\mathcal{O}(1)}$,
- $2^{\mathcal{O}(\mathrm{vi}^2+\mathrm{vi}\cdot\ell)}n^{\mathcal{O}(1)}$,
- $2^{\mathcal{O}(2^{\mathrm{td}}\cdot\ell)}n^{\mathcal{O}(1)}$,

where vc*,* cvd*,* vi*, and* td *denote the vertex cover, cluster vertex deletion number, vertex integrity, and tree-depth of G respectively.*

Given the W[1]-hardness of MaxNDP when parameterized by the feedback vertex number implied by previous works [14, 28], we move on to consider the parameterization by the feedback edge number, and show that it renders the problem tractable.

▶ **Theorem 3** (⋆). *Given an instance $\mathcal{I} = (G, \mathcal{M})$ of* MaxNDP*, there exists an algorithm that computes* $\mathrm{OPT}(\mathcal{I})$ *in time $3^{\mathrm{fes}}n^{\mathcal{O}(1)}$, where* fes *denotes the feedback edge number of G.*

3.2 Approximation Schemes

Using the FPT algorithms of Theorem 2, we develop FPT approximation schemes for MaxNDP when parameterized solely by structural parameters.

▶ **Theorem 4.** *Given an instance $\mathcal{I} = (G, \mathcal{M})$ of* MaxNDP*, one can $(1-\varepsilon)$-approximate* $\mathrm{OPT}(\mathcal{I})$ *in time $2^{\mathcal{O}(\mathrm{cvd}/\varepsilon)}n^{\mathcal{O}(1)}$ and $2^{\mathcal{O}(\mathrm{vi}^2/\varepsilon)}n^{\mathcal{O}(1)}$, where* cvd *and* vi *denote the cluster vertex deletion number and vertex integrity of G respectively.*

Proof. Let $S \subseteq V(G)$ denote the deletion set, which can be computed in time $2^{\mathcal{O}(\mathrm{cvd})}n^{\mathcal{O}(1)}$ for cvd [33] and $2^{\mathcal{O}(\mathrm{vi}\log\mathrm{vi})}n^{\mathcal{O}(1)}$ for vi [13]. Notice that $\mathrm{OPT}(\mathcal{I}[G-S]) \geq \mathrm{OPT}(\mathcal{I}) - |S|$, since every vertex of S can be used to route at most one demand. Consider the case where $\mathrm{OPT}(\mathcal{I}) - |S| \geq (1-\varepsilon)\,\mathrm{OPT}(\mathcal{I}) \iff \mathrm{OPT}(\mathcal{I}) \geq |S|/\varepsilon$. Then, $\mathrm{OPT}(\mathcal{I}[G-S]) \geq (1-\varepsilon)\,\mathrm{OPT}(\mathcal{I})$. Alternatively, it holds that $\mathrm{OPT}(\mathcal{I}) < |S|/\varepsilon$ and the algorithm of Theorem 2 can be used, with $\ell = \mathcal{O}(|S|/\varepsilon)$.

It remains to compute $\mathrm{OPT}(\mathcal{I}[G-S])$. In the case of the cluster vertex deletion set, $G-S$ is a collection of cliques. In that case, one can compute $\mathrm{OPT}(\mathcal{I}[G-S])$ in polynomial time by reducing to an instance of Maximum Matching, on the same vertex set and on

edge set equal to the pairs of $\mathcal{M}$ where both endpoints belong to the same clique. In the case of vertex integrity, it holds that $\mathrm{OPT}(\mathcal{I}[G-S])$ can be computed in FPT time, since it is comprised of $\mathcal{O}(n)$ instances of size at most vi, each of which is solvable in time $2^{\mathcal{O}(\mathrm{vi})}n^{\mathcal{O}(1)}$ due to Theorem 1. ◀

▶ **Theorem 5.** *Given an instance $\mathcal{I}=(G,\mathcal{M})$ of* MaxNDP*, one can $(1-\varepsilon)$-approximate* $\mathrm{OPT}(\mathcal{I})$ *in time $2^{\mathcal{O}(2^{\mathrm{td}}/\varepsilon)}n^{\mathcal{O}(1)}$, where* td *denotes the tree-depth of G.*

Proof. Since G has tree-depth td, one can compute an *elimination forest* of G in time $2^{\mathcal{O}(\mathrm{td}^2)}n^{\mathcal{O}(1)}$ due to [29, 30]. This is a rooted forest on the same vertex set where every pair of vertices adjacent in G adheres to the ancestor/descendant relation. Assume that G is connected, otherwise solve for each connected component. Let r denote the root of the elimination tree, in which case it follows that every connected component of $G-r$ has tree-depth at most $\mathrm{td}-1$. Notice that it holds that $\mathrm{OPT}(\mathcal{I}[G-r])\geq\mathrm{OPT}(\mathcal{I})-1$, since r can be used to route at most one demand. Let $\varepsilon'<\varepsilon$ and consider the case where

$$\mathrm{OPT}(\mathcal{I})-1\geq\frac{1-\varepsilon}{1-\varepsilon'}\cdot\mathrm{OPT}(\mathcal{I})\iff\mathrm{OPT}(\mathcal{I})\geq\frac{1-\varepsilon'}{\varepsilon-\varepsilon'},$$

thus setting $\varepsilon'=\varepsilon/2$ implies that $\mathrm{OPT}(\mathcal{I})\geq\frac{2-\varepsilon}{\varepsilon}$. In this case, it holds that an $(1-\varepsilon')$-approximation of $\mathrm{OPT}(\mathcal{I}[G-r])$ is an $(1-\varepsilon)$-approximation of $\mathrm{OPT}(\mathcal{I})$. On the other hand, if $\mathrm{OPT}(\mathcal{I})<\frac{2-\varepsilon}{\varepsilon}=\mathcal{O}(1/\varepsilon)$, we can use the algorithm of Theorem 2, running in time $2^{\mathcal{O}(2^{\mathrm{td}}/\varepsilon)}n^{\mathcal{O}(1)}$.

Consequently, one can recursively argue about the existence of an approximation scheme, since in the case where a graph has tree-depth equal to 1, the instance is polynomial-time solvable. We proceed with bounding the scheme's running time. Let $T(\mathrm{td},\varepsilon)$ denote the running time for graph G with tree-depth td and error ε, while n' denotes the number of connected components of $G-r$. Notice that it holds that

$$T(\mathrm{td},\varepsilon)\leq\max\left\{2^{\mathcal{O}(2^{\mathrm{td}}/\varepsilon)}n^{\mathcal{O}(1)},n'\cdot T(\mathrm{td}-1,\varepsilon/2)\right\}\leq 2^{\mathcal{O}(2^{\mathrm{td}}/\varepsilon)}n^{\mathcal{O}(1)}+n'\cdot T(\mathrm{td}-1,\varepsilon/2),$$

while the number of the nodes of the same height in the recursion tree is at most n, since each node corresponds to a connected component. Consequently, it holds that

$$T(\mathrm{td},\varepsilon)\leq 2^{\mathcal{O}(2^{\mathrm{td}}/\varepsilon)}n^{\mathcal{O}(1)}+n\cdot\sum_{i=1}^{\mathrm{td}}2^{\mathcal{O}(2^{\mathrm{td}-i}\cdot\frac{1}{\varepsilon/2^i})}n^{\mathcal{O}(1)}=2^{\mathcal{O}(2^{\mathrm{td}}/\varepsilon)}n^{\mathcal{O}(1)}.$$ ◀

4 Inapproximability

Given the FPT approximation scheme of Theorem 5, a natural question arising is whether such an approximation scheme exists for the parameterization by pathwidth as well. Due to the W[1]-hardness of MaxNDP parameterized by $\ell+\mathrm{pw}+\mathrm{fvs}$ [28], there can be no $(1-\varepsilon)$-approximation scheme running in time $f(\mathrm{pw},\mathrm{fvs},\varepsilon)n^{\mathcal{O}(1)}$, yet one of running time $f(\mathrm{pw},\mathrm{fvs},\varepsilon)n^{g(\varepsilon)}$ might be possible.[2] In this section we answer this question in the negative and prove that there exists some constant $0<c'<1$ such that MaxNDP cannot be approximated within a factor of c' in time $f(\mathrm{pw},\mathrm{fvs})n^{\mathcal{O}(1)}$, for any function f, unless the Parameterized Inapproximability Hypothesis [26] fails.

[2] Assuming there existed such an algorithm of running time $f(\mathrm{pw},\mathrm{fvs},\varepsilon)n^{\mathcal{O}(1)}$, setting $\varepsilon<1/\mathrm{OPT}(\mathcal{I})$ results in obtaining a solution of value at least $(1-\varepsilon)\,\mathrm{OPT}(\mathcal{I})>\mathrm{OPT}(\mathcal{I})-1$, i.e., optimal, in time $f(\mathrm{pw},\mathrm{fvs},\mathrm{OPT}(\mathcal{I}))n^{\mathcal{O}(1)}$.

▶ **Theorem 6.** *Assuming the PIH, there exists a constant $c' > 0$ such that* MaxNDP *does not admit a c'-approximation algorithm of running time $f(\mathrm{pw}, \mathrm{fvs})n^{\mathcal{O}(1)}$.*

Proof. Let $\mathcal{I} = (G, k)$ be an instance of Multicolored Densest k-Subgraph, and recall that we assume that G is given to us partitioned into k independent sets $V_1, \ldots, V_k$, where $V_i = \{v_1^i, \ldots, v_n^i\}$. Moreover, let $E^{i_1,i_2} \subseteq E(G)$ denote the edges of G with one endpoint in V_{i_1} and the other in V_{i_2}. For every color class $i \in [k]$, let $s_i = |\{j \in [k] : E^{i,j} \neq \varnothing\}|$, and assume without loss of generality that $1 \leq s_i \leq 3$. Set $\sigma = \sum_{i \in [k]} s_i/2$, and notice that $\frac{k}{2} \leq \sigma \leq \frac{3k}{2}$. Let $\mathrm{OPT}(\mathcal{I}) \leq \sigma$ denote the optimal value of instance $\mathcal{I}$, i.e., the maximum number of edges among the induced subgraphs of G that contain one vertex per color class V_i.

We will construct in polynomial time an instance $\mathcal{J} = (H, \mathcal{M})$ of MaxNDP, where $\mathrm{pw}(H) = \mathcal{O}(k)$, $\mathrm{fvs}(H) = \mathcal{O}(k)$, and $|V(H)| = n^{\mathcal{O}(1)}$, while $\mathcal{M} \subseteq \binom{V(H)}{2}$ is a set of demands. Moreover, it will hold that $\mathrm{OPT}(\mathcal{J}) \leq \ell$, where $\mathrm{OPT}(\mathcal{J})$ denotes the optimal value of instance $\mathcal{J}$, and $\ell = 2k + \sigma$. We will present a reduction such that (i) if $\mathrm{OPT}(\mathcal{I}) = \sigma$, then $\mathrm{OPT}(\mathcal{J}) = \ell$, and (ii) if $\mathrm{OPT}(\mathcal{I}) < c \cdot \sigma$, then $\mathrm{OPT}(\mathcal{J}) < c' \cdot \ell$, for constants c and c' where $0 < c < 1$ and $c' = \frac{9+c}{10}$.

Choice Gadget. For an independent set V_i, we construct the *choice gadget* $\hat{C}_i$ in the following way. First, for $p \in [n]$, we construct paths on vertex sets $\hat{P}_p^i = \{w_q^{i,p} : q \in [3]\}$, as well as paths $\hat{R}_p^i$ on 3 unnamed vertices each. Then, we introduce vertices v_p^i and u_p^i, for $p \in [n+1]$. Next, we add edges $\{u_{n+1}^i, v_1^i\}$ and $\{v_{n+1}^i, u_1^i\}$. Lastly, for $p \in [n]$, we add edges $\{v_p^i, w_1^{i,p}\}$ and $\{w_3^{i,p}, v_{p+1}^i\}$, as well as an edge from u_p^i to one endpoint of $\hat{R}_p^i$, and an edge from u_{p+1}^i to the other endpoint of $\hat{R}_p^i$. See Figure 2 for an illustration. Subsequently, add to $\mathcal{M}$ all pairs (v_p^i, u_{p+1}^i) and (v_p^i, u_{p-1}^i) for $p \in [2, n]$, as well as the pairs (v_1^i, u_2^i) and (v_{n+1}^i, u_n^i). Let $\mathcal{M}_c \subseteq \mathcal{M}$ denote all such pairs added to $\mathcal{M}$ in this step of the construction. Intuitively, we will consider a one-to-one mapping between the vertex v_p^i of V_i being chosen and the vertices of $\hat{P}_p^i$ not being used to route any of the demands in $\mathcal{M}_c$.

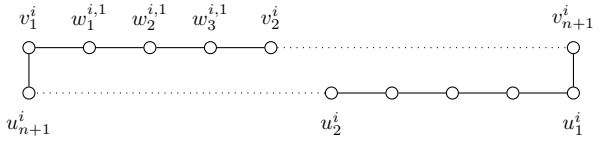


Figure 2 Choice gadget $\hat{C}_i$.

Adjacency vertices. Let $E^{i,j} \neq \varnothing$. Introduce an *adjacency vertex* $e_{i,j}$, and add edges $\{e_{i,j}, w_x^{i,p}\}$ and $\{e_{i,j}, w_y^{j,p}\}$ where $p \in [n]$, and $x, y \in [3]$ such that no other adjacency vertex is adjacent to them (since $s_i \leq 3$ for all $i \in [k]$, there always exist such x and y). If $e = \{v_p^i, v_q^j\} \in E^{i,j}$, then add the pair $(w_x^{i,p}, w_y^{j,q})$ in $\mathcal{M}$. Notice that all adjacency vertices have disjoint neighborhoods, and let $F \subseteq V(H)$ be the set of all adjacency vertices, where $|F| = \sigma$.

This concludes the construction of the instance $\mathcal{J}$. Notice that $H - F$ is a collection of k choice gadgets, each of which is a cycle. Consequently, the graph obtained from $H - F$ by deleting a vertex per choice gadget is a collection of paths, thus both $\mathrm{pw}(H)$ and $\mathrm{fvs}(H)$ are at most $\mathcal{O}(k)$.

We first prove that if $\mathrm{OPT}(\mathcal{I}) = \sigma$, then $\mathrm{OPT}(\mathcal{J}) = \ell$. Consider a function $h\colon [k] \to [n]$ such that $G[\mathcal{V}]$ has σ edges, where $\mathcal{V} = \{v_{h(i)}^i \in V_i : i \in [k]\} \subseteq V(G)$. We construct a family of ℓ vertex-disjoint paths as follows. First, for every $i \in [k]$, we route two demands in $\hat{C}_i$. In particular, we route the demands $(v_{h(i)}^i, u_{h(i)+1}^i)$ as well as $(v_{h(i)+1}^i, u_{h(i)}^i)$, using

the vertices of the shortest path of $\hat{C}_i$ in each case. Note that in this step we have created $2k$ vertex-disjoint paths connecting terminal pairs belonging to $\mathcal{M}_c$, and in every gadget $\hat{C}_i$ the only unused vertices are those of $\hat{P}^i_{h(i)}$ and $\hat{R}^i_{h(i)}$. Then, consider the adjacency vertex $e_{i,j}$, where $i, j \in [k]$. Route the demand $(w^{i,h(i)}_x, w^{j,h(j)}_y)$ via $e_{i,j}$, since both endpoints of the demand are its neighbors and have not been used in any path so far, for some $x, y \in [3]$. Such a demand indeed exists in $\mathcal{M}$, since $\{v^i_{h(i)}, v^j_{h(j)}\} \in E^{i,j}$; if that were not the case then $G[\mathcal{V}]$ has less than σ edges. This procedure results in σ additional demands being routed. Notice that since the neighborhoods of all adjacency vertices are disjoint, the ℓ resulting paths are indeed vertex-disjoint.

It remains to prove that if $\mathrm{OPT}(\mathcal{I}) < c \cdot \sigma$, then $\mathrm{OPT}(\mathcal{J}) < c' \cdot \ell$ (or its contrapositive, as we will do in actuality). We start with Claim 7.

▷ Claim 7. At most 2 demands can be routed in a choice gadget $\hat{C}_i$ in $H - F$, in which case the only unused vertices are those of $\hat{P}^i_p$ and $\hat{R}^i_p$, for some $p \in [n]$. Additionally, it holds that $\mathrm{OPT}(\mathcal{J}) \leq \ell$.

Proof. Let $N = 3n + (n+1)$ and notice that $\hat{C}_i$ is a C_{2N}, thus there are exactly 2 simple paths connecting any pair of its vertices. Moreover, the number of vertices in any simple path between v^i_p and u^i_p, including said endpoints, is $N + 1$. Consequently, to route any demand with both endpoints in $\hat{C}_i$, either $N + 1 - 4$ or $N + 1 + 4$ vertices are used. In case more than 2 demands are routed, at least $3(N + 1 - 4) = 3N - 9$ vertices are used, which is a contradiction since $\hat{C}_i$ consists of $2N$ vertices.

Assume that exactly 2 demands are routed. Moreover, assume that a demand is routed using $N+1+4$ vertices. Then, both routed demands use at least $(N+1+4)+(N+1-4) > 2N$ vertices, which is a contradiction. Therefore, both demands that are routed in $\hat{C}_i$ use the shortest path connecting their endpoints.

Let (v^i_p, u^i_q) and $(v^i_{p'}, u^i_{q'})$ denote the demands routed, where $p < p'$ and $p \in [n]$. It holds that $q = p + 1$, since otherwise $q = p - 1$, and $v^i_{p'}$ belongs to the shortest path connecting (v^i_p, u^i_q), a contradiction. Symmetrically, it follows that $q' = p' - 1$. Lastly, it holds that $q > q'$, since otherwise $u^i_{q'}$ belongs to the shortest path connecting (v^i_p, u^i_q). Consequently, it follows that $q > q' \iff p + 2 > p'$, which implies that $p < p' < p + 2$, i.e., $p' = p + 1$. In that case, the only unused vertices are those of $\hat{P}^i_p$ and $\hat{R}^i_p$.

Notice that $H - F$ is a collection of k choice gadgets. Since in each such gadget at most 2 demands can be routed, it follows that $\mathrm{OPT}(\mathcal{J}) \leq 2k + |F| = 2k + \sigma = \ell$. ◁

We now move on to prove that if $\mathrm{OPT}(\mathcal{J}) \geq c' \cdot \ell$, then $\mathrm{OPT}(\mathcal{I}) \geq c \cdot \sigma$. Let $\mathcal{P}$ denote a collection of $\mathrm{OPT}(\mathcal{J})$ vertex-disjoint paths of H routing demands of $\mathcal{M}$, where $\mathrm{OPT}(\mathcal{J}) \geq c' \cdot \ell$. Additionally, let $\mathcal{C}^{\mathcal{P}}_j$ contain the choice gadgets $\hat{C}_i$ such that there exist *exactly* j paths in $\mathcal{P}$ that route demands in the graph induced by $\hat{C}_i$, for $j \in [0, 2]$. Notice that $(\mathcal{C}^{\mathcal{P}}_0, \mathcal{C}^{\mathcal{P}}_1, \mathcal{C}^{\mathcal{P}}_2)$ defines a partition of the choice gadgets due to Claim 7. Moreover, let $\mathcal{P}^F \subseteq \mathcal{P}$ be comprised of the paths of $\mathcal{P}$ that contain vertices of F. We will say that a path $P \in \mathcal{P}^F$ *intersects* $\hat{C}_i$ if P contains a vertex of $\hat{C}_i$. We define the *loss* of a solution $\mathcal{P}$ to be $L_{\mathcal{P}} = \ell - |\mathcal{P}|$. Notice that due to Claim 7 it holds that $L_{\mathcal{P}} = 2|\mathcal{C}^{\mathcal{P}}_0| + |\mathcal{C}^{\mathcal{P}}_1| + (|F| - |\mathcal{P}^F|)$.

It holds that $\mathrm{OPT}(\mathcal{J}) = |\mathcal{P}| = \ell - L_{\mathcal{P}} \geq c' \cdot \ell = \ell - (1 - c') \cdot \ell$, thus it follows that $L_{\mathcal{P}} \leq (1 - c') \cdot \ell$. Construct a new solution $\mathcal{P}'$ in the following way: for every $\hat{C}_i \in \mathcal{C}^{\mathcal{P}}_0 \cup \mathcal{C}^{\mathcal{P}}_1$, route two demands of $\mathcal{M}_c$ such that there exist two vertex-disjoint paths using only vertices of $\hat{C}_i$. Afterwards, remove any paths of $\mathcal{P}^F$ that are not vertex-disjoint with those. There are at most 3 vertices of F adjacent to vertices of $\hat{C}_i$, and by choosing the routed demands of $\hat{C}_i$ in such a way that at least one of those vertices of $\hat{C}_i$ remains unused, it follows that at most 2

paths of $\mathcal{P}^F$ intersect $\hat{C}_i$. Thus it follows that $L_{\mathcal{P}'} \leq (|F| - |\mathcal{P}^F|) + 2(|\mathcal{C}_0^{\mathcal{P}}| + |\mathcal{C}_1^{\mathcal{P}}|) \leq 2L_{\mathcal{P}}$, therefore $|\mathcal{P}'| = \ell - L_{\mathcal{P}'} \geq \ell - 2L_{\mathcal{P}}$. Furthermore, notice that since for all $i \in [k]$ it holds that $\hat{C}_i \in \mathcal{C}_2^{\mathcal{P}'}$, due to Claim 7 it follows that only the vertices of $\hat{P}^i_{h(i)}$ and $\hat{R}^i_{h(i)}$ remain unused by the paths of $\mathcal{P}'$ that route demands in $\mathcal{M}_c$, for some function $h\colon [k] \to [n]$. Consequently, for any routed demand $(w_x^{i,p}, w_y^{j,q}) \in \mathcal{M} \setminus \mathcal{M}_c$, it holds that $p = h(i)$ and $q = h(j)$.

Let $\mathcal{V} = \{v^i_{h(i)} : i \in [k]\}$, and notice that $|\mathcal{V} \cap V_i| = 1$ for all $i \in [k]$. Let $A = |E(G[\mathcal{V}])|$ denote the number of edges present in the subgraph induced by $\mathcal{V}$. We will prove that $A \geq c \cdot \sigma$, in which case it follows that $\mathrm{OPT}(\mathcal{I}) \geq c \cdot \sigma$. Notice that $A \geq \ell - 2L_{\mathcal{P}} - 2k = \sigma - 2L_{\mathcal{P}}$, since this is the number of routed demands in $\mathcal{M} \setminus \mathcal{M}_c$ by $\mathcal{P}'$, while $(w_x^{i,h(i)}, w_y^{j,h(j)}) \in \mathcal{M}$ implies that $\{v^i_{h(i)}, v^j_{h(j)}\} \in E^{i,j}$.

It suffices to prove that $\sigma - 2L_{\mathcal{P}} \geq c \cdot \sigma$. Since $\sigma - 2L_{\mathcal{P}} \geq \sigma - 2(1-c') \cdot \ell$, we will prove $\sigma - 2(1-c') \cdot \ell \geq c \cdot \sigma$ instead, which is equivalent to $c' \geq 1 + \frac{\sigma}{2\ell}(c-1)$. Since $c - 1 < 0$ and $\frac{\sigma}{2\ell} = \frac{1}{2} \cdot \frac{\sigma}{2k+\sigma} \geq \frac{1}{2} \cdot \frac{k/2}{2k+k/2} = \frac{1}{10}$, the statement holds for $c' = 1 + \frac{1}{10}(c-1) = \frac{9+c}{10}$. ◀

5 XNLP-completeness

The W[1]-hardness results of [14, 28] already imply that MaxNDP is W[1]-hard parameterized by the pathwidth of the input graph. Here we examine in more detail the parameterization solely by pathwidth, and prove that in this case the problem is in fact XNLP-complete. This complexity class was recently brought forth by Bodlaender, Groenland, Nederlof, and Swennenhuis [3] and consists of the parameterized problems such that an instance (x, k), where x can be encoded with n bits and k denotes the parameter, can be solved non-deterministically in time $f(k)n^{\mathcal{O}(1)}$ and space $f(k)\log n$, for some computable function f.

Such a completeness result in fact implies that MaxNDP parameterized by pathwidth is W[t]-hard for all $t \in \mathbb{N}$. To prove said result, we reduce from the XNLP-complete Chained Multicolored Clique, and use a construction quite similar to the one of Theorem 6.

▶ **Theorem 8** (⋆). MaxNDP *parameterized by the pathwidth of the input graph is* XNLP-*complete.*

6 Refining Hardness for Bounded Tree-depth Graphs

In this section, we refine the hardness result of Ene, Mnich, Pilipczuk, and Risteski [14] for bounded tree-depth graphs, by employing a recursive structure introduced in [23]. The reduction of [14] starts from an instance (G, k) of k-Multicolored Clique, and produces an equivalent instance of MaxNDP on a graph of tree-depth, vertex integrity, and feedback vertex number $\mathcal{O}(k^2)$, implying a $n^{o(\sqrt{\mathrm{td}})}$ lower bound under the ETH. We refine their approach, resulting in a reduction that keeps tree-depth linear in k, thereby improving the lower bound to $n^{o(\mathrm{td})}$ under the ETH. As a consequence of our result, it follows that the standard $n^{\mathcal{O}(\mathrm{tw})}$ algorithm for the problem is optimal, even if one considers the class of graphs of bounded tree-depth.

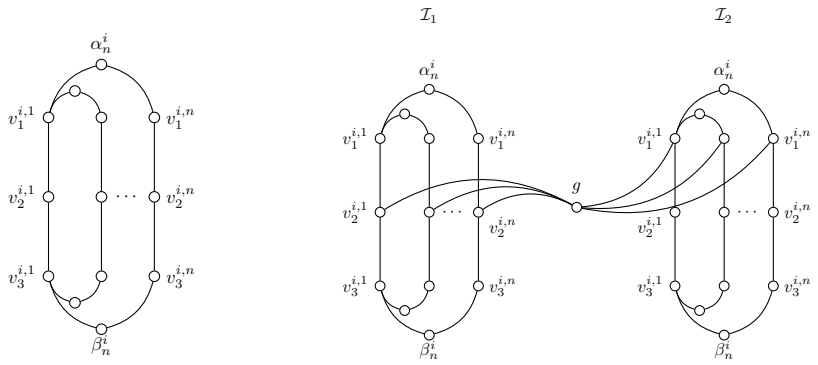
In order to achieve this result, we combine ideas from both [14] and [23]. On a high level, the reduction of [14] consists of k choice gadgets, each of which is used to encode the vertex which is chosen to take part in a supposed clique per color class. Afterwards, it suffices to add $\binom{k}{2}$ vertices in order to verify the existence of edges among all the chosen vertices of the color classes. The deletion of those $\binom{k}{2}$ vertices then gives the bounds for the tree-depth of the graph. In order to avoid this quadratic dependence, we make use of a recursive structure introduced in [23] meant to verify the existence of edges between the chosen vertices, while keeping the tree-depth of the resulting graph linear in k.

▶ **Theorem 9.** *For any computable function f, if there exists an algorithm that solves* MaxNDP *in time $f(\mathrm{td})n^{o(\mathrm{td})}$, where* td *denotes the tree-depth of the input graph, then the ETH is false.*

Proof. Let (G, k) be an instance of k-Multicolored Clique, such that every vertex of G has a self loop, i.e., $\{v, v\} \in E(G)$, for all $v \in V(G)$. Recall that we assume that G is given to us partitioned into k independent sets $V_1, \ldots, V_k$, where $V_i = \{v_1^i, \ldots, v_n^i\}$. Assume without loss of generality that $k = 2^z$, for some $z \in \mathbb{N}$ (one can do so by adding dummy independent sets connected to all the other vertices of the graph). Moreover, let $E^{i_1,i_2} \subseteq E(G)$ denote the edges of G with one endpoint in V_{i_1} and the other in V_{i_2}. We will construct in polynomial time an equivalent instance $(H, \mathcal{M}, \ell)$ of MaxNDP, where H is a graph of tree-depth $\mathrm{td}(H) = \mathcal{O}(k)$, feedback vertex number $\mathrm{fvs}(H) = \mathcal{O}(k^2)$, and size $|V(H)| = n^{\mathcal{O}(1)}$, $\mathcal{M} \subseteq \binom{V(H)}{2}$ is a set of demands, and ℓ is an integer, such that G has a k-clique if and only if at least ℓ demands of $\mathcal{M}$ can be routed in H.

Choice Gadget. For an independent set V_i, we construct the *choice gadget* $\hat{C}_i$ as depicted in Figure 3a. We first construct paths on vertex sets $\hat{P}_j^i = \{v_1^{i,j}, v_2^{i,j}, v_3^{i,j}\}$, where $j \in [n]$. Afterwards, for every $j \in [2, n]$, we introduce vertices α_j^i and β_j^i, connecting them with $v_1^{i,1}, v_1^{i,j}$ and $v_3^{i,1}, v_3^{i,j}$ respectively. Moreover, we add the pair (α_j^i, β_j^i) to $\mathcal{M}$. Intuitively, we will consider a one-to-one mapping between the vertex v_j^i of V_i belonging to a supposed k-clique of G and the vertices of $\hat{P}_j^i$ not being used to route any of the demands added in this step.

Copy Gadget. Given two instances $\mathcal{I}_1$, $\mathcal{I}_2$ of a choice gadget $\hat{C}_i$, when we say that we add a *copy gadget* $(\mathcal{I}_1, \mathcal{I}_2, t)$, where $t \in \{1, 2\}$, we introduce a *copy vertex* g, connect it with all vertices $v_{t+1}^{i,j}$ of $\mathcal{I}_1$ and all vertices $v_1^{i,j}$ of $\mathcal{I}_2$ for $j \in [n]$ and add all the pairs $(v_{t+1}^{i,j}, v_1^{i,j})$ to $\mathcal{M}$.



(a) Choice gadget $\hat{C}_i$. **(b)** Addition of a copy gadget $(\mathcal{I}_1, \mathcal{I}_2, 1)$.

Figure 3 Choice gadgets and how to copy them.

Adjacency Gadget. Let $i_1, i_2, i_1', i_2' \in [k]$. For $i_1 \leq i_2$ and $i_1' \leq i_2'$, we define the *adjacency gadget*[3] $\hat{A}(i_1, i_2, i_1', i_2')$ as follows:

- Consider first the case when $i_1 = i_2 = i$ and $i_1' = i_2' = i'$. Let the adjacency gadget contain instances of the choice gadgets $\hat{C}_i$ and $\hat{C}_{i'}$, as well as a *validation vertex* $m_{i,i'}$. Add edges between $m_{i,i'}$ and all vertices $v_2^{i,j}$ and $v_2^{i',j}$ for $j \in [n]$. If $e = \{v_j^i, v_{j'}^{i'}\} \in E^{i,i'}$, then add the pair $(v_2^{i,j}, v_2^{i',j'})$ to $\mathcal{M}$.

[3] For some high-level intuition regarding the adjacency gadgets, we refer the reader to [23, Section 4].

- Now consider the case when $i_1 < i_2$ and $i'_1 < i'_2$. Then, let $\hat{A}(i_1, i_2, i'_1, i'_2)$ contain instances of choice gadgets $\hat{C}_i$ and $\hat{C}_{i'}$, where $i \in [i_1, i_2]$ and $i' \in [i'_1, i'_2]$, which we will refer to as the *original choice gadgets* of $\hat{A}(i_1, i_2, i'_1, i'_2)$, as well as the adjacency gadgets
  - $\hat{A}\left(i_1, \left\lfloor \frac{i_1+i_2}{2} \right\rfloor, i'_1, \left\lfloor \frac{i'_1+i'_2}{2} \right\rfloor\right)$,
  - $\hat{A}\left(i_1, \left\lfloor \frac{i_1+i_2}{2} \right\rfloor, \left\lceil \frac{i'_1+i'_2}{2} \right\rceil, i'_2\right)$,
  - $\hat{A}\left(\left\lceil \frac{i_1+i_2}{2} \right\rceil, i_2, i'_1, \left\lfloor \frac{i'_1+i'_2}{2} \right\rfloor\right)$,
  - $\hat{A}\left(\left\lceil \frac{i_1+i_2}{2} \right\rceil, i_2, \left\lceil \frac{i'_1+i'_2}{2} \right\rceil, i'_2\right)$.

  Lastly, let $\mathcal{I}$ denote the original choice gadget $\hat{C}_p$, where $p \in [i_1, i_2] \cup [i'_1, i'_2]$. Notice that there are exactly two instances of choice gadget $\hat{C}_p$ appearing as original choice gadgets in the adjacency gadgets just introduced, say instances $\mathcal{I}_1$ and $\mathcal{I}_2$. Add copy gadgets $(\mathcal{I}, \mathcal{I}_1, 1)$ and $(\mathcal{I}, \mathcal{I}_2, 2)$.

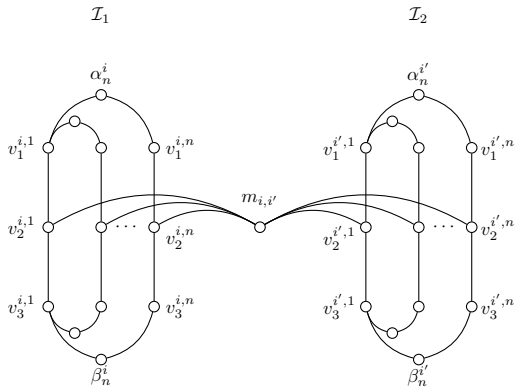


Figure 4 Adjacency gadget $\hat{A}(i, i, i', i')$.

Let graph H be the adjacency gadget $\hat{A}(1, k, 1, k)$ and set $\ell = \gamma \cdot (n-1) + \delta$, where $\gamma = 2k(2k-1)$ and $\delta = 5k^2 - 4k$. Notice that it holds that $|V(H)| = (n \cdot k)^{\mathcal{O}(1)}$ as well as that all copy and validation vertices have disjoint neighborhoods. Additionally, let $\mathcal{M}_{\alpha\beta}$ denote the set of pairs of $\mathcal{M}$ of the form (α_j^i, β_j^i). This concludes the construction of the instance $(H, \mathcal{M}, \ell)$.

▶ **Lemma 10** (⋆). *H has the following properties:*

- *γ is equal to the number of instances of choice gadgets present in H,*
- *δ is equal to the number of copy and validation vertices present in H.*

▶ **Lemma 11** (⋆). *It holds that* $\mathrm{td}(H) = \mathcal{O}(k)$ *and* $\mathrm{fvs}(H) = \mathcal{O}(k^2)$.

▶ **Lemma 12** (⋆). *If G contains a k-clique, then $(H, \mathcal{M}, \ell)$ is a Yes instance of* MaxNDP.

▶ **Lemma 13** (⋆). *If $(H, \mathcal{M}, \ell)$ is a Yes instance of* MaxNDP*, then G contains a k-clique.*

Therefore, in polynomial time, we can construct a graph H, of tree-depth $\mathrm{td} = \mathcal{O}(k)$ and $\mathrm{fvs} = \mathcal{O}(k^2)$ due to Lemma 11, as well as a set of pairs $\mathcal{M}$, such that, due to Lemmas 12 and 13, deciding whether at least ℓ pairs of $\mathcal{M}$ can be routed is equivalent to deciding whether G has a k-clique. ◀

7 Conclusion

In this work we examine in depth Maximum Node-Disjoint Paths under the perspective of parameterized complexity, painting a complete picture regarding its tractability under most standard parameterizations. We additionally employ the use of approximation and present various (in)approximability results, further enhancing our understanding of the problem in the setting of parameterized approximation.

As a direction for future work, we remark that although Theorem 6 excludes the existence of an approximation scheme, a constant-factor approximation algorithm running in time $f(\mathrm{tw})n^{\mathcal{O}(1)}$ remains possible. Regarding the FPT algorithms of Section 3, all but the one of Theorem 3 rely on providing an upper bound on the number of vertices involved in an optimal solution, leading in some cases to even double-exponential running times. The optimality of those running times is thus a natural open question. Lastly, it is unknown whether the problem is FPT parameterized by the cutwidth or the cluster vertex deletion number of the input graph.

References

1 Isolde Adler, Stavros G. Kolliopoulos, Philipp Klaus Krause, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Irrelevant vertices for the planar disjoint paths problem. *J. Comb. Theory, Ser. B*, 122:815–843, 2017. doi:10.1016/J.JCTB.2016.10.001.

2 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995. doi:10.1145/210332.210337.

3 Hans L. Bodlaender, Carla Groenland, Jesper Nederlof, and Céline M. F. Swennenhuis. Parameterized problems complete for nondeterministic FPT time and logarithmic space. *Inf. Comput.*, 300:105195, 2024. doi:10.1016/J.IC.2024.105195.

4 Juhi Chaudhary, Harmender Gahlawat, Michal Wlodarczyk, and Meirav Zehavi. Kernels for the disjoint paths problem on subclasses of chordal graphs. In *18th International Symposium on Parameterized and Exact Computation, IPEC 2023*, volume 285 of *LIPIcs*, pages 10:1–10:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.IPEC.2023.10.

5 Kyungjin Cho, Eunjin Oh, and Seunghyeok Oh. Parameterized algorithm for the disjoint path problem on planar graphs: Exponential in k^2 and linear in n. In *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023*, pages 3734–3758. SIAM, 2023. doi:10.1137/1.9781611977554.ch144.

6 Julia Chuzhoy and David H. K. Kim. On approximating node-disjoint paths in grids. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2015*, volume 40 of *LIPIcs*, pages 187–211. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.APPROX-RANDOM.2015.187.

7 Julia Chuzhoy, David H. K. Kim, and Shi Li. Improved approximation for node-disjoint paths in planar graphs. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016*, pages 556–569. ACM, 2016. doi:10.1145/2897518.2897538.

8 Julia Chuzhoy, David H. K. Kim, and Rachit Nimavat. Improved approximation for node-disjoint paths in grids with sources on the boundary. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018*, volume 107 of *LIPIcs*, pages 38:1–38:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.ICALP.2018.38.

9 Julia Chuzhoy, David H. K. Kim, and Rachit Nimavat. Almost polynomial hardness of node-disjoint paths in grids. *Theory Comput.*, 17:1–57, 2021. doi:10.4086/toc.2021.v017a006.

10 Julia Chuzhoy, David H. K. Kim, and Rachit Nimavat. New hardness results for routing on disjoint paths. *SIAM J. Comput.*, 51(2):17–189, 2022. doi:10.1137/17m1146580.

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 Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate texts in mathematics*. Springer, 2017. doi:10.1007/978-3-662-53622-3.

13 Pål Grønås Drange, Markus S. Dregi, and Pim van 't Hof. On the computational complexity of vertex integrity and component order connectivity. *Algorithmica*, 76(4):1181–1202, 2016. doi:10.1007/S00453-016-0127-X.

14 Alina Ene, Matthias Mnich, Marcin Pilipczuk, and Andrej Risteski. On routing disjoint paths in bounded treewidth graphs. In *15th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2016*, volume 53 of *LIPIcs*, pages 15:1–15:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.SWAT.2016.15.

15 Andreas Emil Feldmann, Karthik C. S., Euiwoong Lee, and Pasin Manurangsi. A survey on approximation in parameterized complexity: Hardness and algorithms. *Algorithms*, 13(6):146, 2020. doi:10.3390/A13060146.

16 Krzysztof Fleszar, Matthias Mnich, and Joachim Spoerhase. New algorithms for maximum disjoint paths based on tree-likeness. *Math. Program.*, 171(1-2):433–461, 2018. doi:10.1007/s10107-017-1199-3.

17 Venkatesan Guruswami, Bingkai Lin, Xuandi Ren, Yican Sun, and Kewen Wu. Parameterized inapproximability hypothesis under exponential time hypothesis. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024*, pages 24–35. ACM, 2024. doi:10.1145/3618260.3649771.

18 Venkatesan Guruswami, Bingkai Lin, Xuandi Ren, Yican Sun, and Kewen Wu. Almost optimal time lower bound for approximating parameterized clique, csp, and more, under ETH. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing, STOC 2025*, pages 2136–2144. ACM, 2025. doi:10.1145/3717823.3718130.

19 Richard M. Karp. On the computational complexity of combinatorial problems. *Networks*, 5(4):45–68, 1975. doi:10.1002/NET.1975.5.1.45.

20 Ken-ichi Kawarabayashi, Yusuke Kobayashi, and Bruce A. Reed. The disjoint paths problem in quadratic time. *J. Comb. Theory, Ser. B*, 102(2):424–435, 2012. doi:10.1016/j.jctb.2011.07.004.

21 Stavros G. Kolliopoulos and Clifford Stein. Approximating disjoint-path problems using packing integer programs. *Math. Program.*, 99(1):63–87, 2004. doi:10.1007/s10107-002-0370-6.

22 Tuukka Korhonen, Michal Pilipczuk, and Giannos Stamoulis. Minor containment and disjoint paths in almost-linear time. In *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024*, pages 53–61. IEEE, 2024. doi:10.1109/FOCS61266.2024.00014.

23 Michael Lampis and Manolis Vasilakis. Structural parameterizations for two bounded degree problems revisited. *ACM Trans. Comput. Theory*, 16(3):17:1–17:51, 2024. doi:10.1145/3665156.

24 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Slightly superexponential parameterized problems. *SIAM J. Comput.*, 47(3):675–702, 2018. doi:10.1137/16M1104834.

25 Daniel Lokshtanov, Pranabendu Misra, Michal Pilipczuk, Saket Saurabh, and Meirav Zehavi. An exponential time parameterized algorithm for planar disjoint paths. *SIAM J. Comput.*, 54(2):321–418, 2025. doi:10.1137/20M1355902.

26 Daniel Lokshtanov, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. Parameterized complexity and approximability of directed odd cycle transversal. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020*, pages 2181–2200. SIAM, 2020. doi:10.1137/1.9781611975994.134.

27 Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. Efficient graph minors theory and parameterized algorithms for (planar) disjoint paths. In *Treewidth, Kernels, and Algorithms - Essays Dedicated to Hans L. Bodlaender on the Occasion of His 60th Birthday*, volume 12160 of *Lecture Notes in Computer Science*, pages 112–128. Springer, 2020. doi:10.1007/978-3-030-42071-0_9.

28 Dániel Marx and Paul Wollan. An exact characterization of tractable demand patterns for maximum disjoint path problems. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015*, pages 642–661. SIAM, 2015. doi:10.1137/1.9781611973730.44.

29 Wojciech Nadara, Michal Pilipczuk, and Marcin Smulewicz. Computing treedepth in polynomial space and linear FPT time. In *30th Annual European Symposium on Algorithms, ESA 2022*, volume 244 of *LIPIcs*, pages 79:1–79:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ESA.2022.79.

30 Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, and Somnath Sikdar. A faster parameterized algorithm for treedepth. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014*, volume 8572 of *Lecture Notes in Computer Science*, pages 931–942. Springer, 2014. doi:10.1007/978-3-662-43948-7_77.

31 Neil Robertson and Paul D. Seymour. Graph minors .XIII. The disjoint paths problem. *J. Comb. Theory, Ser. B*, 63(1):65–110, 1995. doi:10.1006/jctb.1995.1006.

32 Petra Scheffler. *A practical linear time algorithm for disjoint paths in graphs with bounded tree-width.* TU, Fachbereich 3, 1994.

33 Dekel Tsur. Faster parameterized algorithm for cluster vertex deletion. *Theory Comput. Syst.*, 65(2):323–343, 2021. doi:10.1007/S00224-020-10005-W.

34 Michal Wlodarczyk and Meirav Zehavi. Planar disjoint paths, treewidth, and kernels. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023*, pages 649–662. IEEE, 2023. doi:10.1109/FOCS57990.2023.00044.

On the Complexity of Secluded Path Problems

Tesshu Hanaka
Kyushu University, Fukuoka, Japan

Daisuke Tsuru
Kyushu University, Fukuoka, Japan

Abstract

This paper investigates the complexity of finding secluded paths in graphs. We focus on the Short Secluded Path problem and a natural new variant we introduce, Shortest Secluded Path. Formally, given an undirected graph $G = (V, E)$, two vertices $s, t \in V$, and two integers k, l, the Short Secluded Path problem asks whether there exists an s-t path of length at most k with at most l neighbors. This problem is known to be computationally hard: it is W[1]-hard when parameterized by the path length k or by cliquewidth, and para-NP-complete when parameterized by the number l of neighbors. The fixed-parameter tractability is known for $k + l$ or treewidth. In this paper, we expand the parameterized complexity landscape by designing (1) an XP algorithm parameterized by cliquewidth and (2) fixed-parameter algorithms parameterized by neighborhood diversity and twin cover number, respectively. As a byproduct, our results also provide parameterized algorithms for the classic s-t k-Path problem. Furthermore, we introduce the Shortest Secluded Path problem, which seeks a shortest s-t path with the minimum number of neighbors. In contrast to the hardness of the original problem, we reveal that this variant is solvable in polynomial time on unweighted graphs. We complete this by showing that for edge-weighted graphs, the problem becomes W[1]-hard yet remains in XP when parameterized by the shortest path distance between s and t.

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

Keywords and phrases Secluded path, Parameterized complexity, Polynomial-time algorithm

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.4

Funding *Tesshu Hanaka*: JSPS KAKENHI Grant Numbers JP21K17707, JP22H00513, JP25K03077, and JST, CRONOS, Japan Grant Number JPMJCS24K2.

1 Introduction

Path-finding problems are one of the most fundamental graph problems. These problems have been well studied due to their high practicality [1, 3, 7, 9]. However, in many real-world scenarios, simply finding the shortest path is not sufficient. For instance, for secure communication, one may seek a data transmission path for sensitive information that minimizes exposure to potential eavesdroppers. Similarly, in a transportation network, a convoy path that avoids potential attackers could lead to more reliable and secure traveling. In robotics, a path for a robot might need to avoid areas with high sensor noise or potential collisions. These scenarios highlight the need for path-finding problems that consider the *seclusion* of the path from undesirable elements.

Motivated by these applications, the Short Secluded Path problem has been well studied [2, 11, 21, 28]. A vertex subset $U \subseteq V$ is called *l-secluded* if it has at most l neighbors. Formally, Short Secluded Path is defined as follows.

© Tesshu Hanaka and Daisuke Tsuru;
licensed under Creative Commons License CC-BY 4.0

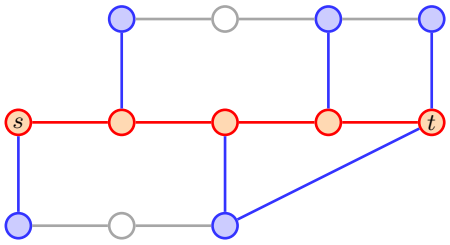


Figure 1 An illustration of a 5-secluded s-t path P of length 5. The s-t path P consists of the red vertices. The blue vertices are neighbors of P.

SHORT SECLUDED PATH

Instance: An undirected graph $G = (V, E)$ where $|V| = n$ and $|E| = m$, two vertices $s, t \in V$, and two integers k, l.

Goal: Determine whether G has an l-secluded s-t path of length at most k.

Figure 1 is an illustration of an l-secluded s-t path of length at most k when $l = 5$ and $k = 5$.[1] Unfortunately, SHORT SECLUDED PATH is NP-complete in general since it is equivalent to s-t HAMILTONIAN PATH if we set $k = n$ and $l = 0$. Due to the NP-hardness of s-t HAMILTONIAN PATH on restricted graph classes [6, 16], SHORT SECLUDED PATH is NP-hard even on planar bipartite graphs of maximum degree 3, strongly chordal split graphs, and chordal bipartite graphs.

Thus, there has been considerable interest in its parameterized complexity. For the natural parameters k and l, Luckow and Fluschnik [21] show that SHORT SECLUDED PATH is W[1]-hard when parameterized by k, whereas it is fixed-parameter tractable (FPT) when parameterized by $k + l$. Note that the problem is para-NP-complete when parameterized by l due to the hardness of s-t HAMILTONIAN PATH. In [28], van Bevern et al. study the fixed-parameter tractability and kernelization complexity of SHORT SECLUDED PATH for structural parameters related to tree-like structures such as treewidth, vertex cover number, feedback vertex set number, and feedback edge set number. For treewidth tw, the authors presented an FPT algorithm for SHORT SECLUDED PATH that runs in $2^{O(\mathrm{tw})}n^{O(1)}$ time. For kernelization, the problem admits a polynomial kernel when parameterized by the combination of k, l, and feedback vertex set number, and also when parameterized by feedback edge set number alone. In contrast, it does not admit a polynomial kernel when parameterized by vertex cover number, by the combination of l and feedback vertex set number, or by the combination of k, l, and treewidth.

1.1 Our contribution

In this paper, we present parameterized algorithms for SHORT SECLUDED PATH with respect to structural parameters related to dense structures such as clique-width, twin cover number, and neighborhood diversity. Our results lead to a more comprehensive understanding of the parameterized complexity of SHORT SECLUDED PATH under structural graph parameters (see Figure 2).

First, we propose an XP algorithm when parameterized by clique-width. Since s-t HAMILTONIAN PATH is known to be W[1]-hard when parameterized by clique-width [12, 16], this XP algorithm provides a tight complexity bound for this parameter. Furthermore,

[1] In this paper, the length of a path is defined as the number of vertices in the path.

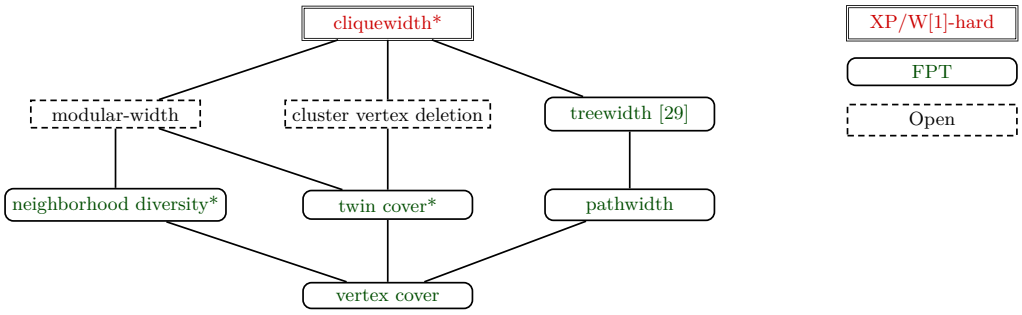


Figure 2 Parameterized complexity of Short Secluded Path with respect to structural graph parameters. The connection between two parameters indicates that the upper parameter p is bounded by some computable function $f(\cdot)$ of the lower parameter q; that is, $p \leq f(q)$. Parameters marked with an asterisk $(*)$ represent our contributions in this paper. Double-bordered rectangles indicate that the parameterized problem belongs to XP but is W[1]-hard; rounded rectangles indicate that the parameterized problem is fixed-parameter tractable (FPT); and dotted rectangles represent cases that remain open.

we design a $\mathrm{nd}(G)^{O(\mathrm{nd}(G)^2)}n^{O(1)}$-time algorithm and a $2^{O(\mathrm{tc}(G)^2)}n^{O(1)}$-time algorithm for Short Secluded Path when parameterized by neighborhood diversity $\mathrm{nd}(G)$ and twin cover number $\mathrm{tc}(G)$, respectively.

Here, it is worth mentioning that our algorithms are designed for Secluded k-Path, which asks whether G has an l-secluded s-t path of length *exactly* k. Thus, by setting $l = n$, our results provide parameterized algorithms for the classical graph problem s-t k-Path. To the best of our knowledge, the parameterized complexity of s-t k-Path with respect to clique-width, twin cover number, and neighborhood diversity was unsettled.

Then we consider the Shortest Secluded Path problem. Unlike Short Secluded Path, the goal of this problem is to find a minimally secluded s-t path among all shortest s-t paths. Formally, Shortest Secluded Path is defined as follows.

Shortest Secluded Path

Instance: An undirected graph $G = (V, E)$ where $|V| = n$ and $|E| = m$, and two vertices $s, t \in V$.

Goal: Find a shortest s-t path P that minimizes the size of its open neighborhood $|N(V(P))|$ in G.

Interestingly, we show that Shortest Secluded Path is solvable in polynomial time on unweighted graphs. This is in contrast to the hardness of Short Secluded Path. However, for positive integer edge-weighted graphs, the problem becomes W[1]-hard when parameterized by the shortest path distance d between s and t. To complement this, we finally present an XP algorithm when parameterized by d.

1.2 Related work

There has been extensive literature on secluded subgraph problems. The study of this area is initiated by Chechik et al. [2], who first introduced the notion of *seclusion* for connectivity problems on graphs. Their original formulation, which they term *exposure*, is defined as the size of the closed neighborhood of a vertex subset. In their seminal work, Chechik et al. consider two problems: Secluded Path and its generalization Secluded Steiner Tree, where the goal is to find a path or a Steiner tree that minimizes its exposure. The authors

show that while Secluded Path is hard to approximate, it is solvable in polynomial time on graphs with bounded degree. For Secluded Steiner Tree, they present a fixed-parameter algorithm parameterized by treewidth. Subsequently, Fomin et al. show that Secluded Steiner Tree is also FPT when parameterized by the exposure of a solution.

A significant shift in perspective came from van Bevern et al. [27], who decouple the solution size from the exposure, and introduce the notion of *l-secludedness*. Within this framework, they investigate the parameterized complexity for finding secluded versions of several fundamental substructures on graphs, such as separators, dominating sets, $\mathcal{F}$-free vertex deletion sets, and independent sets. Building on this work, Golovach et al. [15] show that Connected Secluded $\mathcal{F}$-Free Subgraph is FPT when parameterized by l, and Donkers et al. [8] present a faster FPT algorithm for Secluded Induced Tree parameterized by l. Recently, the concept of seclusion has been extended further, with Mallek et al. [22] studying secluded subgraph problems on directed graphs.

2 Preliminaries

In this paper, we use standard graph-theoretic notations. Let $G = (V, E)$ be an undirected graph, where V is the set of vertices and E is the set of edges. We let $n = |V|$ and $m = |E|$. For a positive integer n, let $[n] := \{1, 2, \cdots, n\}$

For a vertex $v \in V$, its neighborhood is $N_G(v) = \{u \in V \mid \{v, u\} \in E\}$. For a vertex subset $U \subseteq V$, $N_G(U) = \{v \in V \setminus U \mid u \in U, \{u, v\} \in E\}$ denotes the set of neighbors of U. For simplicity, we sometimes use $N(v)$ and $N(U)$ instead of $N_G(v)$ and $N_G(U)$. The subgraph induced by U is denoted by $G[U]$.

The length of a path is defined by the number of vertices on the path. For two vertices $u, v \in V$, the shortest path distance between u and v, denoted by $\mathrm{dist}(u, v)$, is defined as the number of edges (resp., the sum of the weights of edges) in a shortest u-v path on unweighted graphs (resp., edge-weighted graphs). Two distinct vertices u and v are called *twins* if $N(u) \setminus \{v\} = N(v) \setminus \{u\}$. In particular, twins u and v are called *true twins* if the edge $\{u, v\}$ exists, and otherwise called *false twins*.

We assume that the readers are familiar with the basic notions of parameterized complexity [5].

2.1 Cliquewidth

In this subsection, we define the cliquewidth of a graph G. The definition relies on the concept of a k-labeled graph, which we introduce first.

▶ **Definition 1** (k-labeled graph). *Let k be a positive integer. A k-labeled graph is a pair $(G, \mathtt{lab}_G)$ of a graph G and a function $\mathtt{lab}_G : V \to \{1, \cdots, k\}$.*

▶ **Definition 2** (Cliquewidth). *The cliquewidth of a graph G, denoted by $\mathrm{cw}(G)$, is the minimum integer k such that a k-labeled graph $(G, \mathtt{lab}_G)$ can be constructed by repeatedly applying the following operations.*

(O1) *Add a new vertex v with label $i \in [k]$.*

(O2) *Take the disjoint union $(G \oplus H, \mathtt{lab}_{G\oplus H})$ of two k-labeled graphs $(G, \mathtt{lab}_G)$ and $(H, \mathtt{lab}_H)$, with*

$$\mathtt{lab}_{G\oplus H}(v) = \begin{cases} \mathtt{lab}_G(v) & \text{if } v \in V(G), \\ \mathtt{lab}_H(v) & \text{otherwise.} \end{cases}$$

(O3) *Take distinct labels $i, j \in [k]$ for a k-labeled graph G, and add an edge between every pair of vertices labeled by i and by j.*

(O4) *Take distinct labels $i, j \in [k]$ for a k-labeled graph G, and relabel the vertices of label i to label j.*

This construction process for a k-labeled graph can be represented by a rooted binary tree, called a *k-expression tree*. Each node in this tree corresponds to one of the four operations: an *introduce node* (O1), a *union node* (O2), a *join node* (O3), or a *relabel node* (O4). The leaves of a k-expression tree are always introduce nodes, and conversely, all introduce nodes are leaves. The graph associated with any node is the one constructed by the operations in its subtree, and thus the root of the tree represents the final graph G.

A k-expression tree is *irredundant* if for each edge $\{u, v\} \in E(G)$, there is exactly one corresponding join node that adds this edge. Any k-expression tree can be transformed into an irredundant one with $O(n)$ nodes in linear time [4]. Therefore, we can assume without loss of generality that any given k-expression tree is irredundant.

While computing the exact cliquewidth and an optimal expression tree is NP-hard, a polynomial-time approximation algorithm exists. Specifically, a $(2^{\mathrm{cw}(G)+1} - 1)$-expression tree for a graph G with cliquewidth $\mathrm{cw}(G)$ can be computed in $O(n^3)$ time [17, 24, 25].

2.2 Neighborhood diversity and twin cover

A partition $\mathcal{M} = \{M_1, \ldots, M_r\}$ of V is called a *twin partition* of G if every M_i is a set of twins. We call M_i a *module* of $\mathcal{M}$. Then the neighborhood diversity of G is defined as follows.

▶ **Definition 3** (Neighborhood diversity). *The* neighborhood diversity $\mathrm{nd}(G)$ *of G is the minimum number of modules among all twin partitions of G.*

We can compute the *neighborhood diversity* $\mathrm{nd}(G)$ of G and its twin partition in linear time [20, 23, 26]. By definition, each module forms either a clique (if it contains true twins) or an independent set (if it contains false twins).

The *quotient graph* corresponding to a twin partition $\mathcal{M} = \{M_1, M_2, \ldots, M_r\}$ is the graph $Q = (\mathcal{M}, E(\mathcal{M}))$, where $E(\mathcal{M}) = \{\{M_i, M_j\} \mid \exists \{u, v\} \in E,\ u \in M_i,\ v \in M_j\}$. We observe that for any two modules M_i and M_j, either there are no edges between them, or every vertex in M_i is adjacent to every vertex in M_j.

A vertex set $X \subseteq V$ is called a *twin cover* if, for every edge $\{u, v\} \in E$, at least one of the following holds: (i) $u \in X$ or $v \in X$, or (ii) u and v are *true twins* (i.e., adjacent and have identical open neighborhoods). The *twin cover number* of G, denoted $\mathrm{tc}(G)$, is the size of a minimum twin cover of G.

2.3 Integer linear programming

In this subsection, we introduce Integer Linear Programming (ILP) and its fixed-parameter tractability.

▶ **Definition 4** (p-Variable Integer Linear Programming Feasibility (p-ILP)). *Given a matrix $A \in \mathbb{Z}^{m \times p}$ and a vector $b \in \mathbb{Z}^m$, p-Variable Integer Linear Programming Feasibility asks whether there exists a vector $x \in \mathbb{Z}^p$ satisfying $Ax \leq b$.*

It is known that p-ILP is fixed-parameter tractable with respect to the number p of variables.

▶ **Theorem 5** ([13, 18, 19]). *p-Variable Integer Linear Programming Feasibility can be solved in $O(p^{2.5p+o(p)} \cdot L)$ time, where L is the number of bits in the input.*

3 XP Algorithm Parameterized by Cliquewidth

In this section, we design an $n^{2^{O(\mathrm{cw})}}$-time algorithm parameterized by cliquewidth for the problem of determining whether G has an s-t path P of length k with exactly l neighbors. By solving this for each k and l, Secluded k-Path and Short Secluded Path can also be solved in time $n^{2^{O(\mathrm{cw})}}$.

▶ **Theorem 6.** *There is an $n^{2^{O(\mathrm{cw})}}$-time algorithm that, for a graph G of cliquewidth* cw*, determines whether it contains an s-t path of length k with l neighbors.*

Our algorithm is based on dynamic programming on a cliquewidth expression tree of G. It is known that for a graph G with cliquewidth $\mathrm{cw}(G)$, a $(2^{\mathrm{cw}(G)+1}-1)$-expression tree $\mathcal{T}$ with $O(n)$ nodes can be computed in $O(n^3)$ time [17, 24, 25]. For our algorithm, we must distinguish the source s and target t. We achieve this by assigning them two new, unique labels. We can thus assume that we are given an r-expression tree $\mathcal{T}$ with $r \leq 2^{\mathrm{cw}(G)+1}+1$ where s is assigned label $r-1$ and t is assigned label r.

For a given expression tree $\mathcal{T}$, we execute dynamic programming in a bottom-up manner, from the leaves to the root. For each node μ in $\mathcal{T}$, let $G_\mu = (V_\mu, E_\mu)$ be the labeled subgraph associated with μ. The DP table at each node μ stores boolean values for states defined by a tuple of parameters. A state corresponds to the properties of a set $\mathcal{P}$ of vertex-disjoint paths within G_μ, denoted by $\mathcal{P}_\mu$. Intuitively, each path in $\mathcal{P}_\mu$ represents a sub-path of a potential solution in G_μ, i.e., an s-t path of length k with l neighbors. Thus, the DP state needs to track the number of vertices of the solution, the number of their neighborhoods, the number of other vertices, and the number of sub-paths with respect to the labels of the endpoints. The information on endpoint labels of sub-paths is used when merging sub-paths at a join node. We use the notation $V(\mathcal{P}_\mu)$ as a shorthand for $\bigcup_{P\in\mathcal{P}_\mu} V(P)$, the set of all vertices in any path in $\mathcal{P}_\mu$. The parameters in our DP state are defined as follows:

- For each label $i \in \{1, \ldots, r\}$:
  - x_i^μ: the number of vertices in $V(\mathcal{P}_\mu)$ with label i.
  - y_i^μ: the number of vertices in $N(V(\mathcal{P}_\mu)) \cap V_\mu$ with label i.
  - z_i^μ: the number of vertices with label i in V_μ that belong to neither $V(\mathcal{P}_\mu)$ nor its neighborhood.
- For each pair of labels i, j with $1 \leq i \leq j \leq r$:
  - p_{ij}^μ: the number of paths in $\mathcal{P}_\mu$ having one endpoint with label i and the other with label j.

We call $(x_1^\mu, \ldots, x_r^\mu)$ the solution count vector, $(y_1^\mu, \ldots, y_r^\mu)$ the neighborhood count vector, $(z_1^\mu, \ldots, z_r^\mu)$ the remaining vertex count vector, and $(p_{11}^\mu, \ldots, p_{rr}^\mu)$ the path count vector, respectively. The DP entry $\mathrm{DP}_\mu[(x_1^\mu, \ldots, x_r^\mu), (y_1^\mu, \ldots, y_r^\mu), (z_1^\mu, \ldots, z_r^\mu), (p_{11}^\mu, \ldots, p_{rr}^\mu)]$ is a boolean value. It is `true` if and only if there exists a set of vertex-disjoint paths $\mathcal{P}_\mu$ in G_μ that satisfies all of the following conditions:

1. For each label i, the number of vertices in $V(\mathcal{P}_\mu)$ with label i is exactly x_i^μ, i.e., $|V(\mathcal{P}_\mu)| = \sum_{i=1}^r x_i^\mu$.
2. For each label i, the number of vertices in the neighborhood $N(V(\mathcal{P}_\mu)) \cap V_\mu$ with label i is exactly y_i^μ, i.e., $|N(V(\mathcal{P}_\mu))| = \sum_{i=1}^r y_i^\mu$.
3. For each label i, the number of vertices with label i in V_μ that belong to neither $V(\mathcal{P}_\mu)$ nor its neighborhood is exactly z_i^μ, i.e., $|V_\mu \setminus (V(\mathcal{P}_\mu) \cup N(V(\mathcal{P}_\mu)))| = \sum_{i=1}^r z_i^\mu$.
4. For each pair of labels (i, j), the number of paths in $\mathcal{P}_\mu$ with endpoints labeled i and j is exactly p_{ij}^μ.

For convenience, if a set $\mathcal{P}_\mu$ of paths is empty, $|V(\mathcal{P}_\mu)| = 0$. In this case, its neighborhood is also empty, and this corresponds to the state where $x_i^\mu = y_i^\mu = p_{ij}^\mu = 0$ for all i, j.

From these definitions, $0 \leq x_i^\mu, y_i^\mu, z_i^\mu, p_{ij}^\mu \leq n$ for all parameters. Thus, the size of the DP table at each node is bounded by $n^{O(r^2)}$. As initialization, we set all entries of the DP tables to `false`.

The existence of a solution in the original graph G is determined by checking the DP table of the root node, γ. A key observation is that an s-t path of length k with l neighbors exists if and only if there exists a path P whose endpoints are labeled by $r-1$ and r in G_γ satisfying $|V(P)| = k, |N(V(P))| = l$, and $|V_\gamma \setminus (V(P) \cup N(V(P)))| = n-k-l$. Note that s and t have the unique labels $r-1$ and r, respectively. Therefore, an s-t path of length k with l neighbors exists if and only if and only if $\mathrm{DP}_\gamma[(x_1^\gamma, y_1^\gamma, z_1^\gamma), \cdots, (x_r^\gamma, y_r^\gamma, z_r^\gamma), (p_{11}^\gamma, \cdots, p_{rr}^\gamma)] = \texttt{true}$ at the root node γ such that:

- $x_{r-1}^\gamma = x_r^\gamma = 1$,
- $y_{r-1}^\gamma = y_r^\gamma = 0$,
- $z_{r-1}^\gamma = z_r^\gamma = 0$,
- $\sum_{i=1}^r x_i^\gamma = k$,
- $\sum_{i=1}^r y_i^\gamma = l$,
- $\sum_{i=1}^r z_i^\gamma = n-k-l$,
- $p_{(r-1)r}^\gamma = 1$, and
- $p_{ij}^\gamma = 0$ for $i, j (i \leq j)$ except $i = r-1$ and $j = r$.

In the following, we describe the recursive formulas for updating the DP table at each node of the expression tree. The updates for introduce, union, and relabel nodes are relatively straightforward, while the update at a join node is the most challenging part. Intuitively, our DP state tracks, for each label, the number of vertices in the solution, their neighborhoods, and the other vertices, as well as the number of vertex-disjoint paths distinguished by their endpoint labels. For an introduce node, which adds a single labeled vertex, we only need to generate the initial states corresponding to this new vertex, forming a trivial path of length one, or being an isolated vertex not part of the solution. For a union node, which combines two disjoint subgraphs, the new DP table is computed by merging states from its two children. As no edges are added between the subgraphs, paths from each part remain separate. Thus, for any pair of valid states (one from each child), we can compute the resulting state's parameters by simple addition. The update for a relabel node that changes label i to j is also based on addition: for each state, the counts associated with label i are consolidated into the counts for label j.

The main challenge lies with the join node, which adds edges between all vertices of label i and all vertices of label j. These new edges may connect multiple existing paths into longer ones. Therefore, the core of the update is to iterate through all valid states of the child node and, for each state, enumerate all feasible ways that the paths can be connected. Although the number of ways to connect paths can be large, it can be shown that this process can be handled in time $n^{O(r^2)}$. The running time for a join node is determined by iterating through all possible resulting states. Since the size of the DP table is $n^{O(r^2)}$, and we must process each state from the child's table, the update at a join node can be performed in time $n^{O(r^2)}$. As the expression tree has $O(n)$ nodes, the total running time is dominated by the join operations, yielding $n^{O(r^2)} = n^{2^{O(\mathrm{cw}(G))}}$. The details of the algorithm are provided in the appendix.

By applying Theorem 6 for each k, l, we can also solve Secluded k-Path and Short Secluded Path.

▶ **Corollary 7.** *Secluded k-Path and Short Secluded Path can be solved in time $n^{2^{O(\mathrm{cw})}}$.*

4 FPT Algorithm Parameterized by Neighborhood Diversity

In this section, we propose a fixed-parameter algorithm for Short Secluded Path parameterized by neighborhood diversity, which runs in $\mathrm{nd}^{O(\mathrm{nd}^2)}n^{O(1)}$ time. To show this, we first provide a fixed-parameter algorithm for s-t k-Path by neighborhood diversity.

4.1 FPT algorithm for s-t k-Path

We give an ILP formula with $O(\mathrm{nd}^2)$ variables for determining whether G has an s-t path of length k passing through every module.

Let $\mathcal{G}$ be the quotient graph with respect to a twin partition $\mathcal{M} = \{M_1, M_2, \ldots, M_r\}$, where r is the number of modules in the partition of G. Suppose that $M_s = \{s\}$ and $M_t = \{t\}$ for $s, t \in V(G)$ are modules in $\mathcal{M}$, and $\mathcal{G}$ is connected.

We define the ILP instance (P) with variables x_{ij}, x_{ji} (for $\{M_i, M_j\} \in E(\mathcal{G})$), and y_i (for $i \in [r]$) subject to the following constraints.

ILP instance (P)

1. $\sum_{i=1}^{r} y_i = k$

For every $i \in [r]$:

2. a. $\sum_{j \in \{l : M_l \in N_{\mathcal{G}}(M_i)\}} x_{ij} = \sum_{j \in \{l : M_l \in N_{\mathcal{G}}(M_i)\}} x_{ji}$ (if $M_i \neq M_s, M_t$)
   b. $\sum_{j \in \{l : M_l \in N_{\mathcal{G}}(M_s)\}} x_{sj} = 1$
      $\sum_{j \in \{l : M_l \in N_{\mathcal{G}}(M_s)\}} x_{js} = 0$
   c. $\sum_{j \in \{l : M_l \in N_{\mathcal{G}}(M_t)\}} x_{jt} = 1$
      $\sum_{j \in \{l : M_l \in N_{\mathcal{G}}(M_t)\}} x_{tj} = 0$
3. a. $\sum_{j \in \{l : M_l \in N_{\mathcal{G}}(M_i)\}} x_{ij} = y_i$ (if M_i is an independent set)
   b. $1 \leq \sum_{j \in \{l : M_l \in N_{\mathcal{G}}(M_i)\}} x_{ij} \leq y_i$ (if M_i is a clique)

For every partition of $V(\mathcal{G})$ into vertex sets A and B:

4. $\sum_{1 \leq i < j \leq r : \{M_i, M_j\} \in E(\mathcal{G}) \wedge |\{M_i, M_j\} \cap A| = 1} x_{ij} + x_{ji} \geq 1$

For every variable x_{ij}:

5. $x_{ij} \geq 0$

For every variable y_i:

6. $1 \leq y_i \leq |M_i|$

The core idea is to extend the standard *flow-conservation principle* commonly used in network flow problems. We imagine one unit of flow originating at a source module M_s, being consumed at a sink module M_t, and being conserved at all intermediate modules.

The variables x_{ij} and x_{ji} model the flow moving in either direction between two modules M_i and M_j. They also correspond to the number of arcs from M_i to M_j on a directed path P from s to t. This approach captures the path's traversal, potentially back and forth, through the graph of modules. The integer variables y_i track the number of vertices used within each module M_i that the path visits.

The constraint (1) guarantees that the size of a solution is exactly k. The constraints (2) are the low conservation constraints. Constraint (2.a) ensures that for any intermediate module (not a source or sink), the incoming flow equals the outgoing flow, implying that the number of incoming edges and the number of outgoing edges are equal in the intermediate modules. Constraints (2.b) and (2.c) define the source and sink, respectively: exactly one unit of flow leaves the source module M_s, and exactly one unit of flow enters the sink module M_t. Note that $M_s = \{s\}$ and $M_t = \{t\}$ by assumption.

The constraints (3) govern the relationship between the flow passing through a module and the number of vertices used within it (y_i). The constraint (3.a) implies that every incoming edge immediately goes out in an independent set module M. Thus, the number of incoming edges is equal to the number of used vertices in M. The constraint (3.b) implies that P may pass through several vertices and then go out in a clique module M. Thus, the number of incoming edges is at most the number of used vertices in M.

The constraint (4) is the connectivity condition, which ensures that a solution forms a single path. This disallows invalid solutions, such as a subgraph composed of a path and disjoint cycles.

The constraint (5) is a non-negativity constraint. The constraint (6) ensures that for each module, a solution contains at least one vertex in the module and no more vertices than the size of the module.

▶ **Lemma 8.** *The ILP instance* (P) *is feasible if and only if there is an s-t path in G with k vertices that includes at least one vertex from each module $M_i \in \mathcal{M}$.*

From Lemma 8 and Theorem 5, we obtain the following lemma.

▶ **Lemma 9.** *Given a twin partition with r modules, the problem of determining whether there is an s-t path of length k passing through every module can be solved in $r^{O(r^2)}n^{O(1)}$ time.*

4.2 FPT algorithm for Short Secluded Path

Using Lemma 9, we present an FPT algorithm for Secluded k-Path and Short Secluded Path parameterized by neighborhood diversity. A key observation is that if a path visits a module M, then all vertices from this module not on the path are in the neighborhood of the path. Also, all the vertices in modules adjacent to M are in the neighborhood of the path. Therefore, what we only have to do is to solve the problem of determining whether there is an s-t path of length k passing through every guessed module by using Lemma 9.

▶ **Theorem 10.** *Secluded k-Path parameterized by neighborhood diversity* $\mathrm{nd}(G)$ *can be solved in* $\mathrm{nd}(G)^{O(\mathrm{nd}(G)^2)}n^{O(1)}$ *time.*

Proof. First, we compute a twin partition for G with $\mathrm{nd}(G)$ modules in linear time [20,23,26]. We ensure that s and t are in their own singleton modules by splitting the modules containing them. If $s \in M$, we replace M with $M \setminus \{s\}$ and add a new module $M_s = \{s\}$. We do the same for t. This results in a new twin partition $\mathcal{M} = \{M_1, M_2, \ldots, M_r\}$ with $r \leq \mathrm{nd}(G) + 2$. The algorithm proceeds by guessing which modules $\mathcal{M}' \subseteq \mathcal{M}$ contain the vertices of a potential l-secluded s-t path of length at most k. We must always include M_s and M_t in $\mathcal{M}'$. There are $O(2^{r-2}) = O(2^{\mathrm{nd}})$ possible choices for $\mathcal{M}'$.

Let $r' = |\mathcal{M}'|$. For each guess, we check if there exists a valid solution containing at least one vertex in each module in $\mathcal{M}'$ in the subgraph induced by the vertices of $\mathcal{M}'$. If $|r'| > k$ or the induced subgraph is not connected, we can discard this guess.

Let $\mathcal{N} \subseteq \mathcal{M} \setminus \mathcal{M}'$ be the set of modules adjacent to at least one module in $\mathcal{M}'$. For any s-t path P with $V(P) \subseteq \bigcup_{M \in \mathcal{M}'} M$, its neighborhood $N(V(P))$ consists of two parts: (1) all vertices in the modules of $\mathcal{N}$, and (2) all vertices in modules of $\mathcal{M}'$ that are not on the path P. Therefore, $N(V(P)) = (\bigcup_{M \in \mathcal{N}} M) \cup ((\bigcup_{M \in \mathcal{M}'} M) \setminus V(P))$. The size of the neighborhood is $|N(V(P))| = |\bigcup_{M \in \mathcal{N}} M| + |\bigcup_{M \in \mathcal{M}'} M| - |V(P)|$. Since $|V(P)| = k$, $|N(V(P))|$ is given by $|\bigcup_{M \in \mathcal{N}} M| + |\bigcup_{M \in \mathcal{M}'} M| - k$ for a fixed guess $\mathcal{M}'$ (which fixes $\mathcal{N}$). Thus, if $|N(V(P))| > l$, we discard such a guess.

Finally, we only have to determine whether there exists a path of length k in the graph $G[\bigcup_{M' \in \mathcal{M}'} M']$ that visits every module in $\mathcal{M}'$. By Lemma 9, this can be determined in time $r'^{O(r'^2)} n^{O(1)} = \mathrm{nd}^{O(\mathrm{nd}^2)} n^{O(1)}$. We repeat this for all $O(2^{\mathrm{nd}})$ guesses of $\mathcal{M}'$. The total runtime is $2^{\mathrm{nd}} \cdot \mathrm{nd}^{O(\mathrm{nd}^2)} n^{O(1)} = \mathrm{nd}^{O(\mathrm{nd}^2)} n^{O(1)}$. ◀

By applying Theorem 10 to Secluded k'-Path for $2 \leq k' \leq k$, Short Secluded Path can also be solved in $\mathrm{nd}^{O(\mathrm{nd}^2)} n^{O(1)}$ time.

▶ **Corollary 11.** *For a graph G of neighborhood diversity* nd*, Short Secluded Path can be solved in* $\mathrm{nd}^{O(\mathrm{nd}^2)} n^{O(1)}$ *time.*

5 FPT Algorithm Parameterized by Twin Cover Number

In this section, we design a $2^{O(\mathrm{tc}^2)} n^{O(1)}$-time algorithm for Short Secluded Path and Secluded k-Path parameterized by twin cover number.

▶ **Theorem 12.** *Secluded k-Path can be solved in $2^{O(\mathrm{tc}(G)^2)} n^{O(1)}$ time.*

Proof. For any graph G, a twin cover S of size tc can be computed in $O(m + n \cdot \mathrm{tc} + 1.2738^{\mathrm{tc}})$ time [14]. Let $X := S \cup \{s, t\}$ and $\tau := |X| (= \mathrm{tc} + 2)$.

First, we guess the set of edges from $G[X]$ that are part of the secluded path. The number of edges in $G[X]$ is at most $\binom{\tau}{2} \leq \tau^2$. Since a path on τ vertices has at most $\tau - 1$ edges, we guess subsets of at most $\tau - 1$ edges as candidates for partial solutions inside of X. The number of guessed subsets is bounded by $\sum_{i=0}^{\tau-1} \binom{\tau^2}{i} = \tau^{O(\tau)}$.

For each guess, we check if the subgraph formed by these edges consists of a collection of disjoint paths, if s and t are part of the subgraph because they must be endpoints of a path, and if the sum of the number of vertices in the guessed disjoint paths is at most k. This clearly can be done in polynomial time. If these conditions are not satisfied, we discard the current guess.

Let $P_1, P_2, \ldots, P_d$ be sub-paths within $G[X]$ of a guessed partial solution, where $d \leq \tau$. Let $X' = \bigcup_{i=1}^{d} V(P_i)$ be the set of vertices in these paths.

Next, we construct a full s-t path by ordering these sub-paths and connecting them with cliques from $G[V \setminus X]$. First, we guess the orderings of the sub-paths. If s and t belong to the same sub-path P_i, then this must be the entire path. In this case, we check if $d = 1$, $|X'| = |V(P_1)| = k$, and $|N(V(P_1))| \leq l$. If so, P_1 is indeed an l-secluded s-t path of length k, and otherwise we safely conclude that the guess is invalid. If s and t are in different sub-paths, we fix the path containing s at the beginning and the one containing t at the end. We then guess the ordering of the remaining $d - 2$ paths. Since $d - 2 \leq \tau$, the number of such orderings is $(d-2)! = \tau^{O(\tau)}$.

For a fixed ordering of sub-paths, we need to connect them. Since X is a separator and $G[V \setminus X]$ is a union of disjoint cliques, vertices in exactly one clique connect two sub-paths in $G[X]$.

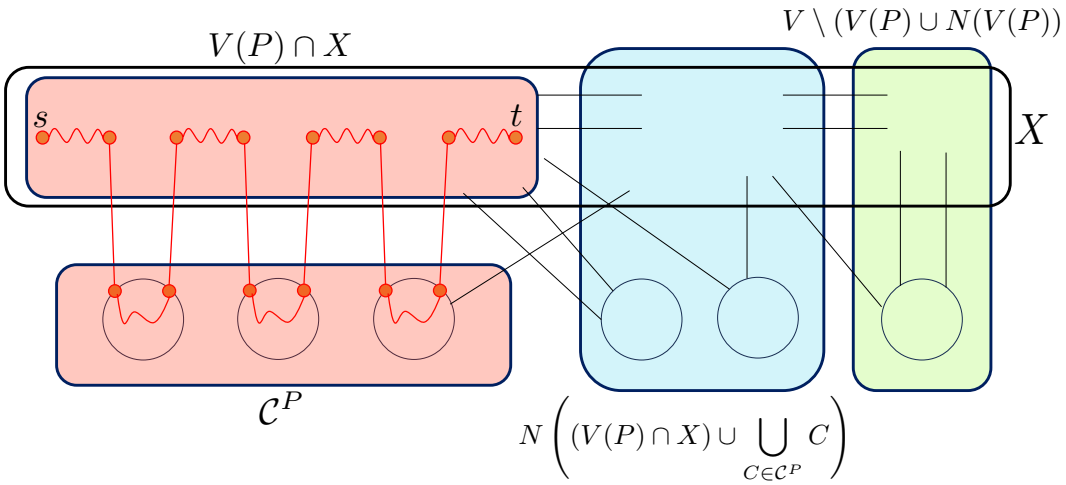


Figure 3 An illustration of a partition of V with respect to a path P in Claim 14.

Here, we categorize cliques in $G[V \setminus X]$ with respect to neighbors in X, and we define the sets of vertices $Y_1, \ldots, Y_q$ where vertices in Y_i have the common neighbors in X, that is, $N(u) \cap X = N(v) \cap X$ for $u, v \in Y_i$. Note that $q \leq 2^\tau$. By the definition of a twin cover, each connected component of $G[V \setminus X]$ is a clique and is fully contained in some Y_i. We say that a clique C in $G[V \setminus X]$ is from Y_i if $V(C) \subseteq Y_i$.

To connect the $d-1$ gaps between the ordered sub-paths, we guess which set Y_j provides the connecting clique for each gap. Since there are $d - 1 = O(\tau)$ gaps and $q = O(2^\tau)$ choices for each, the total number of ways to choose the sequence of Y_j's is $q^{d-1} = 2^{O(\tau^2)}$.

For each such guess, we verify whether it is valid. If a chosen Y_j cannot connect the corresponding sub-paths, we reject it. We also calculate the minimum possible path length. Since we must use at least one vertex from a clique in each of the $d-1$ gaps, the length p of an s-t path constructed is at least $p \geq \sum_{i=1}^{d} |V(P_i)| + d - 1$. If $p > k$, we safely reject it. Furthermore, if the number of times a set Y_i is selected is more than $|Y_i|$, then we reject it because we cannot construct an s-t path even by using all the vertices in Y_i.

Now, for each gap between two sub-paths, we must select a specific clique from the chosen Y_i. Then by the following claim, we can focus on at most r largest cliques in each Y_i.

▷ Claim 13. Suppose that Y_i contains r cliques $C_1, \ldots, C_r$ ordered by non-increasing size ($|C_1| \geq \ldots \geq |C_r|$). Then if there exists an s-t path P that passes through b cliques from Y_i, there exists an s-t path P' that passes through the b largest cliques from Y_i, $C_1, \ldots, C_b$ (or all r cliques if $b \geq r$), such that $|V(P)| = |V(P')|$ and $|N(V(P))| = |N(V(P'))|$.

Now we need to decide vertices to take from the selected cliques. Here, we observe that the vertex set V can be partitioned into four parts: (1) the set $V(P) \cap X$ of vertices on P inside X, (2) the set of vertices $\bigcup_{C \in \mathcal{C}^P} C$ in the selected cliques in $G[V \setminus X]$, (3) the set $N\left((V(P) \cap X) \cup \bigcup_{C \in \mathcal{C}^P} C\right)$ of neighbors of P except for vertices in $\bigcup_{C \in \mathcal{C}^P} C$, and (4) The set $V \setminus V(P) \cup N(V(P))$ of the other vertices which lie outside both X and $N(V(P))$ (see Figure 3). Then the following claim holds.

▷ Claim 14. Let P be an s-t path and let $\mathcal{C}^P$ be the set of cliques in $V \setminus X$ that intersect with $V(P)$. The number of neighbors of the path P is given by:

$$|N(V(P))| = \left|N\left((V(P) \cap X) \cup \bigcup_{C \in \mathcal{C}^P} C\right)\right| + \left|\bigcup_{C \in \mathcal{C}^P} C\right| + |V(P) \cap X| - |V(P)|. \quad (1)$$

If we have fixed the sub-paths in X and the cliques connecting them, we observe that the first three terms $\left|N\left((V(P)\cap X)\cup\bigcup_{C\in\mathcal{C}^P}C\right)\cap V(P)\right|$, $\left|\bigcup_{C\in\mathcal{C}^P}C\right|$, and $|V(P)\cap X|$ in the right-hand side of Equation (1) are determined. Note that $|V(P)\cap X|=|X'|=\sum_{i=1}^{d}|V(P_i)|$. Since $|V(P)|=k$, from Claim 14, $|N(V(P))|$ is also determined.

After fixing the sub-paths in X and the Y_i's for connecting pairs of sub-paths, we can identify the set $\mathcal{C}$ of candidate cliques $\mathcal{C}$ (the largest ones, by Claim 13). Since each clique in Y_i have the same neighbors by the definition of a twin cover, if $|\bigcup_{C\in\mathcal{C}}C|=\sum_{C\in\mathcal{C}}|C|\geq k-|V(P)\cap X|$ is satisfied, we can obtain a path of length k (we just need to arbitrarily choose $k-|V(P)\cap X|$ vertices from selected cliques in $\mathcal{C}$ such that at least one vertex is chosen from each selected clique). This is clearly done in polynomial time.

Finally, we analyze the running time. Guessing the sub-paths in X, the orderings of the sub-paths, and the Y_i's for connecting pairs of sub-paths can be done in time $\tau^{O(\tau)}n^{O(1)}\cdot 2^{O(\tau^2)}n^{O(1)}=2^{O(\tau^2)}n^{O(1)}$. After the guesses, we can check whether each guess is valid by verifying whether $\sum_{C\in\mathcal{C}}|C|\geq k-|V(P)\cap X|$ in polynomial time. Therefore, the total time is $2^{O(\tau^2)}n^{O(1)}=2^{O(\mathrm{tc}^2)}n^{O(1)}$. ◀

▶ **Corollary 15.** *Short Secluded Path can be solved in $2^{O(\mathrm{tc}(G)^2)}n^{O(1)}$ time.*

6 Computational Complexity of Shortest Secluded Path

In this section, we study the computational complexity of Shortest Secluded Path.

6.1 Polynomial-time algorithm on unweighted graphs

In this subsection, we present a polynomial-time algorithm for Shortest Secluded Path on unweighted graphs.

Let $k=\mathrm{dist}(s,t)$ be the distance between s and t. For $i\in\{0,1,\ldots,k\}$, we define the layer $L_i=\{v\in V\mid \mathrm{dist}(s,v)=i \text{ and } \mathrm{dist}(v,t)=k-i\}$ as the set of vertices whose distance from s is i and from t is $k-i$. These layers $L_0,\ldots,L_k$ can be computed in linear time by performing a breadth-first search from s and from t, respectively. By definition, $L_0=\{s\}$ and $L_k=\{t\}$. Let $L=\bigcup_i L_i$ and $R=V\setminus L$. It is easily seen that every vertex v satisfying $\mathrm{dist}(s,v)+\mathrm{dist}(v,t)=k$ is contained in L, and any vertex in R satisfies that $\mathrm{dist}(s,v)+\mathrm{dist}(v,t)>k$. Thus, every shortest s-t path is contained in $G[L]$. Then, the following lemmas hold.

▶ **Lemma 16.** *For any integer $i\in\{0,1,\ldots,k\}$, the neighbors of any vertex in L_i that are also in L must belong to it's layer or adjacent layers. Formally:*

$$N(L_i)\cap L\subseteq L_{i-1}\cup L_{i+1}$$

(Here, we define $L_{-1}=L_{k+1}=\emptyset$.)

Proof. For a vertex $v_i\in L_i$, let $v_j\in L_j(\neq L_i)$ be a neighbor of v_i. By the triangle inequality, we must have $\mathrm{dist}(s,v_i)\leq\mathrm{dist}(s,v_j)+\mathrm{dist}(v_j,v_i)$, which means $i\leq j+1$. Symmetrically, $j\leq i+1$. Together, these imply that j satisfies $|i-j|\leq 1$. ◀

▶ **Lemma 17.** *For any vertex $r\in R$ and any integer $i\in\{0,1,\ldots,k\}$, if r has a neighbor in L_i, then its set of neighbors in L, denoted by $N(r)\cap L$, must satisfy one of the following three conditions: (1) $N(r)\cap L\subseteq L_i$, (2) $N(r)\cap L\subseteq L_{i-1}\cup L_i$, and (3) $N(r)\cap L\subseteq L_i\cup L_{i+1}$.*

From Lemmas 16 and 17, the neighbors of vertices in L_i are in $L_{i-1} \cup L_i \cup L_{i+1} \cup R$. Moreover, the neighbors in R are also adjacent to only $L_{i-1} \cup L_i \cup L_{i+1} \cup R$. This locality is the key to our dynamic programming approach, as it ensures that when extending a path to layer L_{i+1}, the change in the neighborhood of the path only depends on the most recent layers.

Our dynamic programming table, DP, stores values for pairs of adjacent vertices (u_i, u_{i+1}) where $u_i \in L_i$ and $u_{i+1} \in L_{i+1}$. The entry $\mathrm{DP}[u_i, u_{i+1}]$ is defined as the minimum size of the neighborhood of a shortest s-u_{i+1} path that uses the edge $\{u_i, u_{i+1}\}$ as its last edge. If no such path exists, or if $\{u_i, u_{i+1}\} \notin E$, then $\mathrm{DP}[u_i, u_{i+1}] = \infty$. The solution to the overall problem is then $\min_{u_{k-1} \in L_{k-1}} \mathrm{DP}[u_{k-1}, t]$.

For $i = 0$ (the base case), we consider paths of length 2 from s to a vertex $u_1 \in L_1$. Since $\mathrm{dist}(s, u_1) = 1$, the edge $\{s, u_1\}$ must exist. The path consists of edge $\{s, u_1\}$. The size of its neighborhood is $|(N(s) \cup N(u_1)) \setminus \{s, u_1\}|$. Thus, we initialize $\mathrm{DP}[s, u_1] = |N(s) \cup N(u_1)| - 2$.

For $1 \leq i < k$, the recursive formula for $\mathrm{DP}[u_i, u_{i+1}]$ is computed as follows:

$$\mathrm{DP}[u_i, u_{i+1}] = \begin{cases} \infty & \text{if } \{u_i, u_{i+1}\} \notin E, \\ \min\limits_{u_{i-1} \in L_{i-1}} \{\mathrm{DP}[u_{i-1}, u_i] + |N(u_{i+1}) \setminus (N(u_{i-1}) \cup N(u_i))|\} & \text{otherwise.} \end{cases}$$

Here, $|N(u_{i+1}) \setminus (N(u_{i-1}) \cup N(u_i))|$ represents the number of new neighbors added when extending the path from u_i to u_{i+1}. Due to the locality shown by Lemmas 16 and 17, u_{i+1} never shares neighbors of $\bigcup_{j=0}^{i-2} L_j$. Thus, these new neighbors are precisely those in $N(u_{i+1})$ that are not already neighbors of the path ending in $\{u_{i-1}, u_i\}$.

Finally, we analyze the running time. The sets L_i can be computed in $O(m + n)$ time. The size of the DP table is $O(n^2)$ since the number of layers is at most n. For three vertices u, v, w, $|N(u) \cap (N(v) \cup N(w))|$ can be computed in polynomial time in advance. Thus, each entry $\mathrm{DP}[u_i, u_{i+1}]$ can be computed in $O(n)$ time from $\mathrm{DP}[u_{i-1}, u_i]$. Therefore, the total running time is $O(n^3)$.

▶ **Theorem 18.** *Shortest Secluded Path can be solved in $O(n^3)$ time.*

6.2 Shortest Secluded Path on weighted graphs

In this subsection, we discuss Shortest Secluded Path on edge-weighted graphs. We assume that each edge weight is a positive integer. We first show Shortest Secluded Path on weighted graphs is W[1]-hard with respect to the shortest path distance between s and t by a reduction from Multicolored Clique. Here, the shortest path distance between s and t is defined by the minimum weight of an s-t path.

Multicolored Clique

Instance: A graph $G = (V_1 \cup \cdots \cup V_k, E)$ with n vertices where each V_i forms an independent set.

Goal: Determine whether G has a clique of size k.

Multicolored Clique is W[1]-hard when parameterized by k even on regular graphs [10].

▶ **Theorem 19.** *Shortest Secluded Path on edge-weighted graphs is W[1]-hard parameterized by the shortest path distance d between s and t in the input graph.*

Proof. We show the parameterized reduction from Multicolored Clique to Shortest Secluded Path.

Let an r-regular graph $G = (V, E)$ be an instance of Multicolored Clique, where $V = V_1 \cup V_2 \cup \cdots \cup V_k$. Then, we construct an equivalent instance $G' = (V', E', s, t)$ of Shortest Secluded Path on edge-weighted graphs. We first define $V' = V \cup \{s, t\} \cup V_E \cup W$ where $V_E = \{v_e \mid e \in E\}$ and $W = \{w_h \mid h \in [k-1]\}$. Then we connect s to all vertices in V_1 and t to all vertices in V_k with edges of weight 1. Moreover, for each $h \in [k-1]$, we connect w_h to all vertices in $V_h \cup V_{h+1}$ with edges of weight 1. Finally, for $e = \{u, v\} \in E$, we connect v_e to u and v with edges of weight $k+1$. Figure 4 illustrates the constructed graph G'.

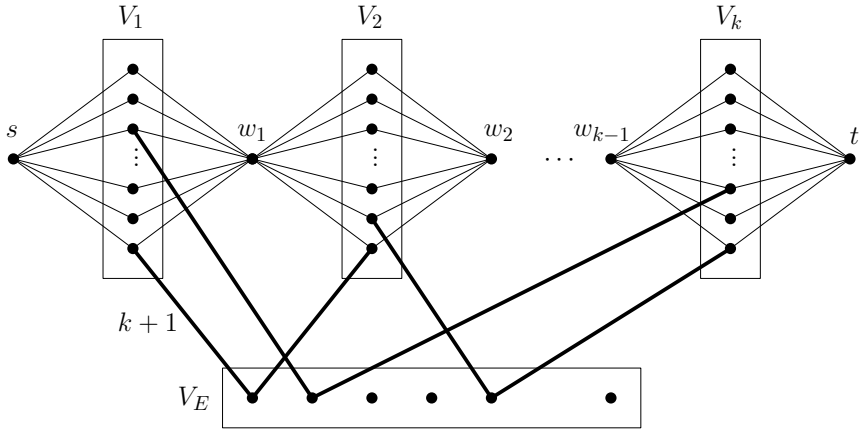


Figure 4 The reduction from Multicolored Clique to Shortest Secluded Path in Theorem 19. The weight of each thin edge is 1 and the weight of each bold edge is $k+1$.

Clearly, G' can be constructed in polynomial time. By the construction of G', the shortest path distance between s and t is $2k$. In the following, we show that there is a clique of size k in G if and only there is a shortest s-t path P of weight $2k$ satisfying $|N(V'(P))| \leq rk - \binom{k}{2} + n - k$ in G' where $V'(P)$ is the set of vertices of P on G'.

Let $C = \{v_1, v_2, \ldots, v_k\}$ be a clique of size k in G, where $v_i \in V_i$. We define an s-t path $P = \langle s, v_1, w_1, v_2, w_2, \ldots, w_{k-1}, v_k, t \rangle$ of weight $2k$. Since all edges in $G'[V' \setminus V_E]$ have weights 1, the weight of P is $2k$, and thus P is a shortest s-t path in G'.

We next show P satisfies $|N(V'(P))| \leq rk - \binom{k}{2} + n - k$. We observe that $V \setminus V'(P) \subseteq N(V'(P))$ holds. Since $|V'(P) \cap V| = k$, the neighborhood $N(V'(P))$ of P includes $n - k$ vertices in $V \setminus V'(P)$. Furthermore, since G is an r-regular graph, any vertex in V has r neighbors in V_E in G'. Since C is a clique in G, for $1 \leq i < j \leq k$, any pair $v_i, v_j \in C$ has an edge $\{v_i, v_j\}$. Thus, v_i and v_j have a common neighbor $v_{\{v_i, v_j\}} \in V_E$. Therefore, P has only $rk - \binom{k}{2}$ neighbors in V_E. Consequently, $|N(V'(P))| = rk - \binom{k}{2} + n - k$ holds.

Conversely, suppose that there is a shortest s-t path P of weight $2k$ satisfying $|N(V'(P))| \leq rk - \binom{k}{2} + n - k$ in G'. If P contains any vertex in V_E, the weight of P is at least $2k + 2 > 2k$. Thus, P does not contain any vertex in V_E. Hence, by the construction of G', P must pass through $w_1, \cdots, w_{k-1}$ and exactly one vertex in each $V_1, \cdots, V_k$, and $V \setminus V(P) \subseteq N(V'(P))$ holds. Consequently, $|N(V(P)) \setminus V_E| = n - k$ and P has at most $rk - \binom{k}{2}$ neighbors in V_E. Since every vertex in $V'(P) \cap V$ has r neighbors in V_E, they must share $\binom{k}{2}$ vertices in V_E. Since $|V'(P) \cap V| = k$, every pair of $u, v \in V'(P) \cap V$ must share $v_{\{u,v\}} \in V_E$, which implies that there exists edge $\{u, v\}$ in G. This means that $V'(P) \cap V$ forms a clique of size k, which completes the proof. ◀

Finally, we present an XP algorithm for Shortest Secluded Path that runs in $n^{O(d)}$ time where d is the shortest path distance between s and t in the input graph.

▶ **Theorem 20.** *Shortest Secluded Path on edge-weighted graphs can be solved in $n^{O(d)}$ time, where d is the shortest path distance between s and t in the input graph.*

Proof. Given an input graph $G = (V, E)$, we first compute the shortest path distance $d = \mathrm{dist}(s, t)$ between s and t by Dijkstra's algorithm [7]. Since the edge weights are nonnegative integers, any shortest s-t path contains at most d edges. Thus, all shortest s-t paths can be enumerated in $n^{O(d)}$ time. Since the neighborhood of a shortest s-t path can be computed in polynomial time, Shortest Secluded Path can be solved in $n^{O(d)}$ time. ◀

7 Conclusion

In this paper, we investigated the structural parameterizations of Short Secluded Path and the computational complexity of Shortest Secluded Path.

For the structural parameterizations of Short Secluded Path, we presented an XP algorithm parameterized by cliquewidth and FPT algorithms parameterized by neighborhood diversity and twin cover number, respectively. Regarding Shortest Secluded Path, we first proposed a polynomial-time algorithm for unweighted graphs. For edge-weighted graphs, however, we proved that the problem is W[1]-hard but is in XP when parameterized by the shortest path distance between s and t.

A natural future direction is to investigate the parameterized complexity of Short Secluded Path with respect to other structural parameters, such as the cluster vertex deletion number and modular-width. Furthermore, improving the running times of our algorithms is an interesting challenge. In particular, given an r-expression tree, our XP algorithm parameterized by cliquewidth runs in $n^{O(r^2)}$ time. It is worth considering whether this running time can be improved to $n^{O(r)}$. It would also be interesting to design a single-exponential FPT algorithm parameterized by the twin cover number, since our current algorithm runs in $2^{O(\mathrm{tc}(G)^2)}n^{O(1)}$ time.

References

1. R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.
2. Shiri Chechik, Matthew P. Johnson, Merav Parter, and David Peleg. Secluded connectivity problems. *Algorithmica*, 79(3):708–741, 2017. doi:10.1007/S00453-016-0222-Z.
3. Boris V. Cherkassky, Andrew V. Goldberg, and Tomasz Radzik. Shortest paths algorithms: Theory and experimental evaluation. *Math. Program.*, 73(2):129–174, May 1996. doi:10.1007/BF02592101.
4. Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discret. Appl. Math.*, 101(1-3):77–114, 2000. doi:10.1016/S0166-218X(99)00184-5.
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. Alexsander Andrade de Melo, Celina M. H. de Figueiredo, and Uéverton S. Souza. On the computational difficulty of the terminal connection problem. *RAIRO Theor. Informatics Appl.*, 57:3, 2023. doi:10.1051/ITA/2023002.
7. Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959. doi:10.1007/BF01386390.
8. Huib Donkers, Bart M. P. Jansen, and Jari J. H. de Kroon. Finding k-secluded trees faster. *J. Comput. Syst. Sci.*, 138:103461, 2023. doi:10.1016/J.JCSS.2023.05.006.
9. David Eppstein. Finding the k shortest paths. *SIAM J. Comput.*, 28(2):652–673, 1998. doi:10.1137/S0097539795290477.

10 Michael R. Fellows, Danny Hermelin, Frances A. Rosamond, and Stéphane Vialette. On the parameterized complexity of multiple-interval graph problems. *Theor. Comput. Sci.*, 410(1):53–61, 2009. doi:10.1016/J.TCS.2008.09.065.

11 Fedor V. Fomin, Petr A. Golovach, Nikolay Karpov, and Alexander S. Kulikov. Parameterized complexity of secluded connectivity problems. *Theory Comput. Syst.*, 61(3):795–819, 2017. doi:10.1007/S00224-016-9717-X.

12 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Intractability of clique-width parameterizations. *SIAM J. Comput.*, 39(5):1941–1956, 2010. doi:10.1137/080742270.

13 András Frank and Éva Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Comb.*, 7(1):49–65, 1987. doi:10.1007/BF02579200.

14 Robert Ganian. Improving vertex cover as a graph parameter. *Discret. Math. Theor. Comput. Sci.*, 17(2):77–100, 2015. doi:10.46298/DMTCS.2136.

15 Petr A. Golovach, Pinar Heggernes, Paloma T. Lima, and Pedro Montealegre. Finding connected secluded subgraphs. *J. Comput. Syst. Sci.*, 113:101–124, 2020. doi:10.1016/J.JCSS.2020.05.006.

16 Tesshu Hanaka and Yasuaki Kobayashi. Finding a minimum spanning tree with a small non-terminal set. *Theor. Comput. Sci.*, 1033:115092, 2025. doi:10.1016/J.TCS.2025.115092.

17 Petr Hlinený and Sang-il Oum. Finding branch-decompositions and rank-decompositions. *SIAM J. Comput.*, 38(3):1012–1032, 2008. doi:10.1137/070685920.

18 Hendrik W. Lenstra Jr. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 8(4):538–548, 1983. doi:10.1287/MOOR.8.4.538.

19 Ravi Kannan. Minkowski's convex body theorem and integer programming. *Math. Oper. Res.*, 12(3):415–440, 1987. doi:10.1287/MOOR.12.3.415.

20 Michael Lampis. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica*, 64(1):19–37, 2012. doi:10.1007/S00453-011-9554-X.

21 Max-Jonathan Luckow and Till Fluschnik. On the computational complexity of length- and neighborhood-constrained path problems. *Inf. Process. Lett.*, 156:105913, 2020. doi:10.1016/J.IPL.2019.105913.

22 Nadym Mallek, Jonas Schmidt, and Shaily Verma. A parameterized study of secluded structures in directed graphs. *CoRR*, abs/2502.06048, 2025. doi:10.48550/arXiv.2502.06048.

23 Ross M. McConnell and Jeremy P. Spinrad. Modular decomposition and transitive orientation. *Discret. Math.*, 201(1-3):189–241, 1999. doi:10.1016/S0012-365X(98)00319-7.

24 Sang-il Oum. Approximating rank-width and clique-width quickly. *ACM Trans. Algorithms*, 5(1):10:1–10:20, 2008. doi:10.1145/1435375.1435385.

25 Sang-il Oum and Paul D. Seymour. Approximating clique-width and branch-width. *J. Comb. Theory B*, 96(4):514–528, 2006. doi:10.1016/J.JCTB.2005.10.006.

26 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.

27 René van Bevern, Till Fluschnik, George B. Mertzios, Hendrik Molter, Manuel Sorge, and Ondrej Suchý. The parameterized complexity of finding secluded solutions to some classical optimization problems on graphs. *Discret. Optim.*, 30:20–50, 2018. doi:10.1016/J.DISOPT.2018.05.002.

28 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.

Kernelization for H-Coloring

Yael Berkman
The Academic College of Tel Aviv-Yaffo, Tel Aviv, Israel

Ishay Haviv
The Academic College of Tel Aviv-Yaffo, Tel Aviv, Israel

Abstract

For a fixed graph H, the H-Coloring problem asks whether a given graph admits an edge-preserving function from its vertex set to that of H. A seminal theorem of Hell and Nešetřil asserts that the H-Coloring problem is NP-hard whenever H is loopless and non-bipartite. A result of Jansen and Pieterse implies that for every graph H, the H-Coloring problem parameterized by the vertex cover number k admits a kernel with $O(k^{\Delta(H)})$ vertices and bit-size bounded by $O(k^{\Delta(H)} \cdot \log k)$, where $\Delta(H)$ denotes the maximum degree in H. For the case where H is a complete graph on at least three vertices, this kernel size nearly matches conditional lower bounds established by Jansen and Kratsch and by Jansen and Pieterse.

This paper presents new upper and lower bounds on the kernel size of H-Coloring problems parameterized by the vertex cover number. The upper bounds arise from two kernelization algorithms. The first is purely combinatorial, and its size is governed by a structural quantity of the graph H, called the non-adjacency witness number. As applications, we obtain kernels whose size is bounded by a fixed polynomial for natural classes of graphs H with unbounded maximum degree, such as planar graphs and, more broadly, graphs with bounded degeneracy. More strikingly, we show that for almost every graph H, the degree of the polynomial that bounds the size of our combinatorial kernel grows only logarithmically in $\Delta(H)$. Our second kernel leverages linear-algebraic tools and involves the notion of faithful independent representations of graphs. It strengthens the general bound from prior work and, among other applications, yields near-optimal kernels for problems concerning the dimension of orthogonal graph representations over finite fields. We complement our kernelization results with conditional lower bounds, thereby nearly settling the kernel complexity of the problem for various target graphs H.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Mathematics of computing → Graph coloring; Mathematics of computing → Combinatorial algorithms

Keywords and phrases Kernelization, Graph coloring, Graph homomorphism

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.5

Related Version *Full Version*: `http://arxiv.org/abs/2507.13129`

Funding *Yael Berkman*: Research supported by the Israel Science Foundation (grant No. 1218/20).
Ishay Haviv: Research supported by the Israel Science Foundation (grant No. 1218/20).

Acknowledgements We thank the anonymous reviewers for their valuable feedback.

1 Introduction

A central concept in graph theory is that of *graph homomorphisms*, namely, functions from the vertex set of one graph G to the vertex set of another graph H that map adjacent vertices in G to adjacent vertices in H. If such a function exists, the graph G is said to be *H-colorable*. For a fixed graph H, the computational H-Coloring problem asks whether a given input graph is H-colorable. One may always assume that the target graph H is a core, i.e., a graph admitting no homomorphism to a proper subgraph, as replacing H with a minimal core subgraph does not change the problem. This class of problems occupies a fundamental place

© Yael Berkman and Ishay Haviv;
licensed under Creative Commons License CC-BY 4.0

in computational graph theory and has been studied extensively for decades (see, e.g., [21]). A milestone in the area is the dichotomy theorem of Hell and Nešetřil [20], which asserts that the problem is solvable in polynomial time whenever H has a loop or is bipartite, and is NP-hard otherwise. In recent years, H-Coloring problems have received attention from multiple perspectives, such as algorithmic design [8, 41, 39], computational lower bounds [5], and fine-grained complexity [35, 37, 10].

This paper investigates the family of H-Coloring problems from the perspective of parameterized complexity, more specifically, from the viewpoint of kernelization. Parameterized complexity is a framework for analyzing decision problems whose instances are equipped with a quantitative parameter, with the goal of determining the effect of the parameter's value on the problem's computational complexity. A central theme in this field, known as kernelization or data reduction, seeks to design efficient preprocessing algorithms that substantially reduce the input size. These algorithms, called *kernels*, take an input instance (x, k), where x is the main input and k is the parameter, and transform it in polynomial time into an equivalent instance (x', k') whose bit-size is bounded by $f(k)$ for a computable function $f : \mathbb{N} \to \mathbb{N}$ depending only on k. The slowest achievable growth rate of such a function f reflects the problem's compressibility limits with respect to the parameter k. In the context of graph problems, a widely-studied and ubiquitous parameter is the size of a vertex cover, i.e., a set of vertices that includes at least one endpoint of every edge in the graph. When H-Coloring is parameterized by the vertex cover number k, the input consists of a graph G along with a vertex cover of G of size k.

An archetypal problem within the H-Coloring framework is the q-Coloring problem for a fixed integer q, which has attracted persistent interest in algorithmic and complexity-theoretic research. This problem asks whether the vertices of a given graph can be colored with q colors, so that no two adjacent vertices share a color. It precisely coincides with the H-Coloring problem where H is the complete graph on q vertices, and is well known to be NP-hard for $q \geq 3$. The kernelization complexity of the q-Coloring problem parameterized by the vertex cover number k was first comprehensively studied by Jansen and Kratsch in [23] (see also [22]). They showed that for every integer $q \geq 3$, the problem admits a kernel producing graphs with $O(k^q)$ vertices which can be encoded in $O(k^q)$ bits. This result was subsequently refined by Jansen and Pieterse [24], who obtained a kernel with $O(k^{q-1})$ vertices and bit-size $O(k^{q-1} \cdot \log k)$. Remarkably, this bound is nearly optimal, as it was proved in [23, 24] that for every integer $q \geq 3$ and any real $\varepsilon > 0$, the problem does not admit a kernel of size $O(k^{q-1-\varepsilon})$ unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$, a containment that would imply the collapse of the polynomial-time hierarchy to its third level [42].

Another captivating case of the H-Coloring problem arises from a geometric lens on graph theory, namely, through the concept of *orthogonal representations* introduced by Lovász [28] in 1979. For a positive integer d and a field $\mathbb{F}$, a d-dimensional orthogonal representation of a graph $G = (V, E)$ over $\mathbb{F}$ assigns to each vertex $v \in V$ a non-self-orthogonal vector $x_v \in \mathbb{F}^d$, such that for every edge $\{u, v\} \in E$, the vectors x_u and x_v are orthogonal. Here, two vectors $x, y \in \mathbb{F}^d$ are called orthogonal if their standard inner product $\langle x, y \rangle = \sum_{i=1}^{d} x_i y_i$ equals zero, and a vector $x \in \mathbb{F}^d$ is self-orthogonal if $\langle x, x \rangle = 0$. The orthogonal representation is termed *faithful* if two vertices u and v are adjacent in G if and only if their vectors x_u and x_v are orthogonal. The minimum dimension d for which a graph admits a (faithful) orthogonal representation over $\mathbb{F}$ captures an intriguing structural property, and has found applications in combinatorics, theoretical computer science, and information theory (see [29, Chapter 10] and, e.g., [30, 4, 2, 12, 17, 11]). This motivates the study of the d-Ortho-Dim$_{\mathbb{F}}$ problem, which, for fixed d and $\mathbb{F}$, asks whether a given graph

admits a d-dimensional orthogonal representation over $\mathbb{F}$. This problem naturally fits within the H-Coloring framework by letting $H = H(\mathbb{F}, d)$ be the (possibly infinite) graph whose vertices are all non-self-orthogonal vectors in $\mathbb{F}^d$, with adjacency defined by orthogonality over $\mathbb{F}$. Note that the problem is known to be NP-hard for every integer $d \geq 3$ and every field $\mathbb{F}$ [36]. It was recently shown [18] that for every integer $d \geq 3$ and every field $\mathbb{F}$, the d-Ortho-Dim$_{\mathbb{F}}$ problem parameterized by the vertex cover number k admits a kernel with $O(k^d)$ vertices and bit-size $O(k^d)$, whereas for any $\varepsilon > 0$, it admits no kernel of size $O(k^{d-1-\varepsilon})$ unless NP $\subseteq$ coNP/poly. A sharper upper bound was obtained in [18] for the real field $\mathbb{R}$, yielding a near-optimal kernel with $O(k^{d-1})$ vertices and bit-size $O(k^{d-1} \cdot \log k)$. For finite fields, however, the precise kernel complexity remained unresolved, with a multiplicative gap of roughly k separating the upper and lower bounds on the kernel size.

The near-optimal kernels for the q-Coloring problems were extended by Jansen and Pieterse in [24] to the H-Coloring problem for arbitrary graphs H. They showed that for every graph H, the H-Coloring problem parameterized by the vertex cover number k admits a kernel with $O(k^{\Delta(H)})$ vertices and bit-size $O(k^{\Delta(H)} \cdot \log k)$, where $\Delta(H)$ denotes the maximum degree of a vertex in H. This result was actually established in a stronger form, with respect to the less restrictive parameter known as the twin cover number. While the bound on the kernel size from [24] is essentially optimal when H is a complete graph, it is natural to ask to what extent the dependence on the maximum degree of H captures the kernel complexity of the problem for general graphs H. This question serves as the driving force behind our work, which explores the polynomial behavior of the kernel complexity of H-Coloring problems parameterized by the vertex cover number.

1.1 Our Contribution

The present paper provides upper and lower bounds on the kernel size of H-Coloring problems parameterized by the vertex cover number. Our upper bounds include two kernelization algorithms: the first is purely combinatorial, and the second leverages linear-algebraic tools. Below, we elaborate on each of them. Table 1 provides an overview of their applications.

The Combinatorial Kernel

Our first kernel is simple and combinatorial, inspired by the strategy developed for coloring problems by Jansen and Kratsch in [23]. Consider an instance of the H-Coloring problem parameterized by the vertex cover number k, namely, a graph $G = (V, E)$ and a vertex cover $X \subseteq V$ of G of size k. For a fixed integer q, the kernel starts with the subgraph of G induced by X, and then, for every set $S \subseteq X$ of size at most q whose vertices share a neighbor from $V \setminus X$ in G, introduces a new vertex whose neighborhood is exactly S. While the number of vertices in the constructed graph is clearly bounded by $O(k^q)$, we prove that the correctness of the kernelization for H-Coloring is governed by a structural quantity of the graph H, which we denote by $q(H)$ and refer to as the *non-adjacency witness number* of H.

Given a set of vertices in H that have no common neighbor, one may ask whether this can be certified by a succinct witness, namely, a small subset that also has no common neighbor. The non-adjacency witness number $q(H)$ is defined as the smallest size of such a subset, whose existence is guaranteed for every set of vertices in H with no common neighbor (see Definition 6). This notion has previously been studied in the literature, with particular attention given to graphs H with $q(H) = 2$ (see, e.g., [6, 14, 13]). We note that $q(H)$ may also be formulated as the smallest integer q for which the collection of open neighborhoods in H satisfies what is known as the Helly property of order q. Our analysis reveals that

the size of the aforementioned kernel for the H-Coloring problem parameterized by the vertex cover number is bounded by a polynomial, whose degree is given by the non-adjacency witness number of H.

▶ **Theorem 1.** *For every graph H, the H-Coloring problem parameterized by the vertex cover number k admits a kernel with $O(k^{q(H)})$ vertices and bit-size $O(k^{q(H)})$.*

We remark that when applying Theorem 1, it is sometimes useful to replace H with a core subgraph of H. As noted earlier, this substitution does not change the problem, however, it may reduce the non-adjacency witness number and thereby lead to a smaller kernel.

In an attempt to exploit Theorem 1 and to compare it with the kernel of $O(k^{\Delta(H)})$ vertices obtained in [24], we undertake a systematic study of the non-adjacency witness number of graphs. Our analysis illuminates connections to fundamental graph parameters, such as maximum degree, clique number, and degeneracy, as well as structural properties related to the presence of specific subgraphs. By combining these insights with Theorem 1, we obtain economical kernels for a broad range of graphs H. In particular, as demonstrated below, we achieve kernels whose size is bounded by a fixed polynomial for natural classes of graphs H with unbounded maximum degree.

For a fixed integer d, consider the class of d-degenerate graphs, namely, those graphs in which every subgraph has a vertex of degree at most d. We show that for every such graph H, its non-adjacency witness number satisfies $q(H) \leq d + 1$, so Theorem 1 implies that the corresponding H-Coloring problem parameterized by the vertex cover number k admits a kernel whose size is bounded by $O(k^{d+1})$. For planar graphs, which are known to be 5-degenerate, this yields a bound of $O(k^6)$ on the kernel size, and via a refined analysis, we further reduce it to $O(k^4)$. Perhaps most strikingly, we apply Theorem 1 to show that for *almost every graph* H, there is a kernel for the H-Coloring problem parameterized by the vertex cover number, where the degree of the polynomial bounding its size grows only logarithmically with the maximum degree of H. To this end, we provide a tight estimate for the typical non-adjacency witness number of random graphs.

The Algebraic Kernel

Our second kernel for the H-Coloring problem parameterized by the vertex cover number is more algebraic in nature. For certain graphs H, it allows us to sharpen the kernel size from Theorem 1 by a multiplicative linear factor. Somewhat surprisingly, the condition that makes this improvement applicable is related to the concept of faithful orthogonal representations of graphs (which, as described earlier, assign a non-self-orthogonal vector to each vertex, such that two vertices are adjacent if and only if their vectors are orthogonal). Concretely, we show that if a graph H admits a faithful d-dimensional orthogonal representation over some computationally efficient field, then the H-Coloring problem parameterized by the vertex cover number k admits a kernel with $O(k^{d-1})$ vertices and bit-size $O(k^{d-1} \cdot \log k)$.

In fact, we obtain this kernel size under a weaker assumption on the graph H, using the concept of *independent representations* of graphs. This notion refers to an assignment of a d-dimensional vector to each vertex of H, so that the vector associated with each vertex does not lie in the vector space spanned by the vectors of its neighborhood. Note that every orthogonal representation is, in particular, an independent representation. The minimum dimension of an independent representation of a graph over a given field characterizes a well-studied graph quantity, called *minrank*, which plays a pivotal role in the study of fundamental problems in information theory, such as Shannon capacity [15], index coding [1, 40, 16], storage capacity [33], and hat guessing games [38]. For our purposes, we introduce a *faithful*

analogue of this notion, demanding that two vertices u and v in the graph are adjacent if and only if the vector associated with u lies in the linear subspace spanned by the vectors of the neighborhood of v (see Definition 9). With this faithful variant in hand, we prove the following theorem (see also Theorem 14).

▶ **Theorem 2** (Simplified). *If a graph H has a faithful d-dimensional independent representation over either a finite field or the real field $\mathbb{R}$, then the H-Coloring problem parameterized by the vertex cover number k admits a kernel with $O(k^{d-1})$ vertices and bit-size $O(k^{d-1} \cdot \log k)$.*

A few remarks are in order here. First, it is not difficult to show that every graph H has a faithful independent representation of dimension $\Delta(H)+1$ over any sufficiently large field (see Lemma 13). By Theorem 2, this implies that for every graph H, the H-Coloring problem parameterized by the vertex cover number k admits a kernel with $O(k^{\Delta(H)})$ vertices and bit-size $O(k^{\Delta(H)} \cdot \log k)$. Therefore, Theorem 2 strengthens the kernel bound implied by [24]. It is worth noting that a result of Maehara and Rödl [31] asserts that every graph H has a faithful orthogonal representation over $\mathbb{R}$ of dimension $2 \cdot \Delta(H)$, and it is an open question whether this dimension can be reduced to $\Delta(H) + 1$ (see, e.g., [29, Chapter 10.4]). Thus, for Theorem 2 to fully subsume the kernel bound of [24], it is somewhat crucial to adopt the more flexible notion of faithful independent representations (rather than orthogonal).

Second, we observe that for every graph H that possesses a faithful d-dimensional independent representation over some field, the non-adjacency witness number satisfies $q(H) \leq d$ (see Lemma 11). This shows that Theorem 2 improves the kernel size obtained via Theorem 1 only when the minimum dimension of a faithful independent representation of H matches the non-adjacency witness number of H (in fact, of a minimal core subgraph of H). This situation arises, for example, for the aforementioned graphs $H = H(\mathbb{F}, d)$, which correspond to the existence of a d-dimensional orthogonal representation over a field $\mathbb{F}$. As an application of Theorem 2, we unleash a near-optimal kernel for the d-Ortho-Dim$_{\mathbb{F}}$ problems over finite fields $\mathbb{F}$, thereby settling a question posed in [18].

▶ **Theorem 3.** *For every integer $d \geq 3$ and every finite field $\mathbb{F}$, the d-Ortho-Dim$_{\mathbb{F}}$ problem parameterized by the vertex cover number k admits a kernel with $O(k^{d-1})$ vertices and bit-size $O(k^{d-1} \cdot \log k)$.*

We further demonstrate the applicability of Theorem 2 for the celebrated family of *Kneser graphs*. For positive integers m and r with $m \geq 2r$, the Kneser graph $K(m, r)$ has all r-subsets of an m-element set as vertices, where two are adjacent if they are disjoint. It turns out that the non-adjacency witness number of $K(m, r)$ is $m - 2r + 2$, coinciding with its chromatic number that was determined in an influential paper of Lovász [27]. By Theorem 1, this shows that the $K(m, r)$-Coloring problem parameterized by the vertex cover number k admits a kernel with $O(k^{m-2r+2})$ vertices and bit-size. To further reduce this kernel size, we prove that $K(m, r)$ admits a faithful $(m - 2r + 2)$-dimensional independent representation over any sufficiently large field, which yields, via Theorem 2, a kernel with $O(k^{m-2r+1})$ vertices and bit-size $O(k^{m-2r+1} \cdot \log k)$. In particular, for the 3-chromatic Kneser graphs $K(2r + 1, r)$, which include the iconic Petersen graph $K(5, 2)$, the resulting kernel has near-quadratic size (see Figure 1).

The proof of Theorem 2 builds on a powerful sparsification technique that was introduced by Jansen and Pieterse in [25] and applied in their work [24]. To illustrate this technique, consider an instance of the H-Coloring problem parameterized by the vertex cover number, consisting of a graph $G = (V, E)$ and a vertex cover $X \subseteq V$ of G. The approach begins by introducing variables over some field $\mathbb{F}$ that represent an assignment of vertices from H to

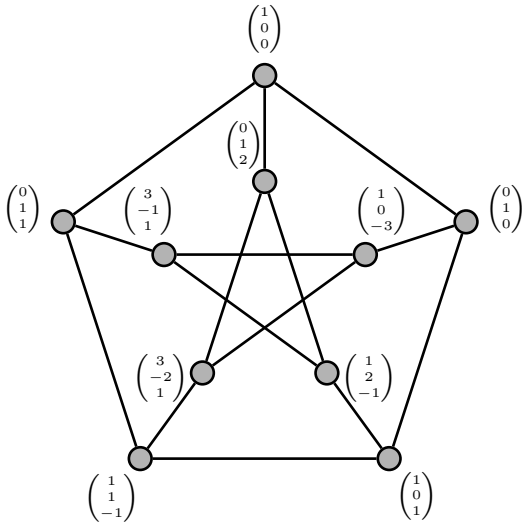


Figure 1 The Petersen graph – A faithful 3-dimensional orthogonal representation over $\mathbb{R}$.

the vertices in X. Then, for every vertex $v \in V \setminus X$, one constructs one or more low-degree polynomials in the variables corresponding to its neighbors in X, which evaluate to zero if and only if v can be assigned a compatible vertex from H. The crux is that it suffices to retain in the graph G only a subset of the vertices in $V \setminus X$, whose corresponding polynomials linearly span all the others. The degree of the used polynomials determines the dimension of the entire space of polynomials, and consequently, the resulting kernel size. This method was applied in [24], where the assignment of each vertex of X is encoded by a set of variables representing an indicator vector over the binary field $\mathbb{F}_2$.

The way we apply the machinery of [24, 25] to prove Theorem 2 is substantially different. We rely on the existence of a faithful d-dimensional independent representation of the graph H over some field $\mathbb{F}$, interpreting its vectors as encodings of the vertices of H. Accordingly, we associate each vertex in the vertex cover X with a d-dimensional variable vector over $\mathbb{F}$ to represent its assignment. The kernelization algorithm first applies our combinatorial kernel to ensure that every vertex outside X has degree at most d. Then, it aims to reduce the number of vertices outside X that have degree exactly d. By the definition of an independent representation, a vertex v outside X can be assigned a compatible vertex from H only if every set of d vectors of its neighbors in X is linearly dependent over $\mathbb{F}$. This condition can be naturally captured by requiring the determinant polynomial over the variables corresponding to such d neighbors of v to be zero. Once the vectors of the neighbors of v span a subspace of dimension smaller than d, the assignment of a compatible vertex to v is essentially realized through some other vertex outside X of degree at most $d-1$.

However, the degree of the determinant polynomial of a $d \times d$ matrix is d. In order to obtain polynomials of degree $d-1$, which in turn yield a kernel with $O(k^{d-1})$ vertices, we use an idea from [18, 19] and require all vectors in the faithful independent representation of H to share a fixed value, say 1, in their first entry. When the field $\mathbb{F}$ is sufficiently large relative to the number of vertices in H, this can be achieved by applying an invertible linear transformation to the vectors in the representation to ensure that all first entries are nonzero, and then scaling them appropriately. Over a smaller field, though, such a representation may not exist. We overcome this obstacle by showing that if a graph has a faithful d-dimensional independent representation over some field $\mathbb{F}$, then it also has one in which all vectors have 1

as their first entry, over any sufficiently large *extension* field of $\mathbb{F}$ (see Lemma 12). This allows us to apply the approach described above over such an extension field and obtain a kernel of the desired size.

Table 1 Kernel size for the H-Coloring problem parameterized by the vertex cover number k for various target graphs H.

Graph H	Kernel bit-size	Kernel type
d-degenerate	$O(k^{d+1})$	combinatorial
Planar	$O(k^4)$	combinatorial
Odd cycle C_{2m+1}, $m \geq 2$	$O(k^2)$	combinatorial
$H(\mathbb{F}, d)$, $\mathbb{F}$ finite, $d \geq 3$	$\widetilde{O}(k^{d-1})$	algebraic
Kneser graph $K(m, r)$	$\widetilde{O}(k^{m-2r+1})$	algebraic

Lower Bounds

Given our combinatorial and algebraic kernels for the H-Coloring problem parameterized by the vertex cover number k, it is natural to ask how close the obtained kernel sizes come to being optimal. As already discussed, and shown in [23, 24], when H is the complete graph on q vertices with $q \geq 3$, the problem admits no kernel of size $O(k^{q-1-\varepsilon})$ for any $\varepsilon > 0$, unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$. This firmly settles the kernel complexity for all cases where H, or its minimal core subgraph, is a complete graph. Turning to other fundamental graphs, the first non-trivial case that arises is that of odd cycles of length at least five. For every integer $m \geq 2$, it is easy to verify that the cycle C_{2m+1} on $2m+1$ vertices has non-adjacency witness number $q(C_{2m+1}) = 2$. Consequently, Theorem 1 yields a kernel for the C_{2m+1}-Coloring problem parameterized by the vertex cover number k with $O(k^2)$ vertices and bit-size. Furthermore, neither Theorem 2 nor the kernel from [24] improves upon this bound. Another notable case is the family of 3-chromatic Kneser graphs $K(2r+1, r)$, for which Theorem 2 yields a kernel with $O(k^2)$ vertices and bit-size $O(k^2 \cdot \log k)$. The following theorem indicates that, in both cases, the obtained kernel sizes are nearly optimal.

▶ **Theorem 4.** *For each of the following cases, for any real number $\varepsilon > 0$, the H-Coloring problem parameterized by the vertex cover number k does not admit a kernel of size $O(k^{2-\varepsilon})$ unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.*

1. *$H = C_{2m+1}$ for an integer $m \geq 1$.*
2. *$H = K(2r+1, r)$ for an integer $r \geq 1$.*

We prove Theorem 4 in a stronger form in several respects. First, we show that the near-quadratic lower bound persists even when the H-Coloring problem is parameterized by the total number of vertices in the input graph. Since the entire vertex set of a graph trivially forms a vertex cover, this directly implies the statement above. Second, the proof encompasses not only the graphs H stated in the theorem but also extends to various other graphs H. Specifically, the result is proved for all non-bipartite core graphs that are *projective*, a class that is known to cover nearly all graphs, including prominent and well-studied graph families (see, e.g., [26]). Third, as is standard in such results, the lower bound applies to the compressibility of the H-Coloring problem into any target problem, not necessarily itself. Our generalized result is stated as follows.

▶ **Theorem 5.** *For every non-bipartite projective core graph H and for any real number $\varepsilon > 0$, the H-Coloring problem parameterized by the number of vertices n does not admit a compression of size $O(n^{2-\varepsilon})$ unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.*

The proof of Theorem 5 is based on a compression lower bound, due to Chen, Jansen, Okrasa, Pieterse, and Rzążewski [3], for a variant of the H-Coloring problem called List-H-Coloring. In this variant, the input consists of a graph G along with a list of allowed vertices of H for each vertex of G, and the goal is to decide whether there exists a homomorphism from G to H that respects these lists. This problem generalizes the standard H-Coloring problem, which corresponds to the special case where all lists are equal to the full vertex set of H. Feder, Hell, and Huang [7] introduced a class of geometric intersection graphs, called *bi-arc graphs*, and used this class to fully characterize the complexity of the List-H-Coloring problem. They showed that the problem is solvable in polynomial time if H is bi-arc, and NP-hard otherwise. Chen et al. [3] proved that, unless NP $\subseteq$ coNP/poly, the List-H-Coloring problem parameterized by the number of vertices admits no subquadratic compression for any H that is not bi-arc. They also posed the question of whether such a lower bound holds for the (non-list) H-Coloring problem, when H is loopless and non-bipartite. As a step toward this challenge, Theorem 5 confirms the lower bound for the substantial class of non-bipartite projective core graphs H. To this end, we show that for such graphs H, instances of List-H-Coloring can be efficiently reduced into instances of H-Coloring, with the number of vertices preserved up to a multiplicative constant. The proof borrows a gadget construction due to Okrasa and Rzążewski [35] and echoes early NP-hardness proofs for H-Coloring problems [34, 32], as well as recent lower bounds in fine-grained complexity [35, 37].

When the H-Coloring problems are parameterized by the number of vertices, they admit a trivial kernel of quadratic size. Therefore, the lower bounds implied by Theorem 5 for the vertex cover parameterization are limited to the near-quadratic regime. To surpass this barrier, we establish a lower bound on the kernel size of the H-Coloring problem parameterized by the vertex cover number for all non-bipartite projective core graphs H with $q(H) \geq 4$. We show that, for each such graph H, the degree of any polynomial that bounds the kernel size is unlikely to be smaller than the non-adjacency witness number of H minus one. The precise statement and proof appear in the full version of the paper. Note that this lower bound is reasonably close to our upper bounds. Indeed, for every non-bipartite projective core graph H with $q(H) \geq 4$, the upper bound given in Theorem 1 exceeds this lower bound by only a near-linear multiplicative factor. Since almost all graphs are non-bipartite projective cores, our results determine the kernel complexity of the problem up to a near-linear factor for almost every graph H. Furthermore, if a non-bipartite projective core graph H admits a faithful independent representation of dimension $q(H) \geq 4$ over a computationally efficient field, then the kernel given in Theorem 2 essentially meets the lower bound (see also Theorem 14). As an illustrative consequence, we derive that the kernel sizes we achieve for all non-bipartite Kneser graphs are nearly tight (extending Item 2 of Theorem 4).

1.2 Outline

The rest of the paper is organized as follows. In Section 2, we collect several definitions and facts that will be used throughout the paper. In Section 3, we present and analyze our combinatorial and algebraic kernels for the H-Coloring problem parameterized by the vertex cover number, thereby proving Theorems 1 and 2. Further results, including lower bounds on compression size, the study of the non-adjacency witness number, and detailed applications to specific graphs H, appear in the full version of the paper. Concluding remarks and open questions appear in the final Section 4.

2 Preliminaries

Throughout the paper, we omit floor and ceiling signs when they are not crucial, and all logarithms are in base 2. For a positive integer n, we write $[n]$ to denote the set $\{1, 2, \ldots, n\}$.

2.1 Graphs

The graphs considered in this paper are, unless stated otherwise, finite and simple, meaning they have no loops or parallel edges. For a graph $G = (V, E)$ and a set $X \subseteq V$, we let $G[X]$ denote the subgraph of G induced by X. The set X is called a *vertex cover* when the graph $G[V \setminus X]$ is edgeless. For a vertex $v \in V$, we let $N_G(v)$ denote the set of neighbors of v in G. As is customary, we denote by $\Delta(G)$ the maximum degree of a vertex in G. For two graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$, a *homomorphism* from G to H is a mapping $f : V_G \to V_H$ such that for all vertices $u, v \in V_G$ with $\{u, v\} \in E_G$, it holds that $\{f(u), f(v)\} \in E_H$. If there exists a homomorphism from G to H, the graph G is said to be *H-colorable*.

2.2 Linear Algebra

For a field $\mathbb{F}$ and an integer d, two vectors $x, y \in \mathbb{F}^d$ are said to be *orthogonal* if $\langle x, y \rangle = 0$, with respect to the standard inner product defined by $\langle x, y \rangle = \sum_{i=1}^{d} x_i y_i$, where all operations are performed over $\mathbb{F}$. A vector $x \in \mathbb{F}^d$ is *self-orthogonal* if $\langle x, x \rangle = 0$, and *non-self-orthogonal* otherwise.

The *order* of a finite field is the number of its elements. It is well known that the order of any finite field is a prime power, and that every prime power is the order of some finite field. For two fields $\mathbb{F}$ and $\mathbb{K}$, we say that $\mathbb{K}$ is an *extension field* of $\mathbb{F}$ if $\mathbb{F} \subseteq \mathbb{K}$ and the operations of $\mathbb{F}$ agree with those of $\mathbb{K}$ when restricted to elements in $\mathbb{F}$. A finite field of order p^m, for a prime p and a positive integer m, has an extension field of order $p^{m'}$ for every integer $m' \geq m$.

A multivariate polynomial over a field $\mathbb{F}$ is called *homogeneous of degree d* if each of its monomials has degree d, and is called *multilinear* if each of them forms a product of distinct variables. The set of all multilinear homogeneous polynomials of degree d in n variables over a field $\mathbb{F}$ forms a vector space over $\mathbb{F}$ of dimension $\binom{n}{d}$.

2.3 Parameterized Complexity

We collect here basic definitions on kernelization from the field of parameterized complexity. For an in-depth introduction to the topic, the reader is referred to, e.g., [9]. A *parameterized problem* is a set $Q \subseteq \Sigma^* \times \mathbb{N}$ for some finite alphabet Σ. A *compression* (also known as generalized kernel) for a parameterized problem $Q \subseteq \Sigma^* \times \mathbb{N}$ into a parameterized problem $Q' \subseteq \Sigma^* \times \mathbb{N}$ is an algorithm that given an instance $(x, k) \in \Sigma^* \times \mathbb{N}$ returns in time polynomial in $|x| + k$ an instance $(x', k') \in \Sigma^* \times \mathbb{N}$, such that $(x, k) \in Q$ if and only if $(x', k') \in Q'$, and in addition, $|x'| + k' \leq h(k)$ for some computable function h. The function h is called the *size* of the compression. If $|\Sigma| = 2$, the function h is called the *bit-size* of the compression. When we say that a parameterized problem Q admits a compression of size h, we mean that there exists a compression of size h for Q into *some* parameterized problem. A compression for a parameterized problem Q into itself is called a *kernelization*, or simply a *kernel*, for Q.

For a fixed graph H, the computational H-Coloring problem asks to decide whether an input graph G is H-colorable. When parameterized by the vertex cover number, the input further includes a vertex cover X of G, whose size is the parameter of the problem. While

this definition is standard and convenient, we note that including the vertex cover X in the input is not essential, as X can be replaced by a vertex cover of G computed via an efficient 2-approximation algorithm.

3 Kernels for H-Coloring

In this section, we present and analyze our kernelization algorithms for the H-Coloring problem parameterized by the vertex cover number. We begin with a combinatorial kernel, whose size is governed by the non-adjacency witness number of H, and then proceed to an algebraic kernel, whose size is bounded in terms of the dimension of a faithful independent representation of H.

3.1 The Combinatorial Kernel

The non-adjacency witness number of a graph measures the smallest number of vertices needed to certify that a given set of vertices has no common neighbor. We formally define it below and study it in detail in the full version of the paper.

▶ **Definition 6.** *The* non-adjacency witness number *of a graph $G = (V, E)$, denoted $q(G)$, is the smallest positive integer q such that for every set $T \subseteq V$ of vertices with no common neighbor in G, there exists a set $T' \subseteq T$ of size $|T'| \leq q$ with no common neighbor in G.*

A main ingredient of our combinatorial kernel is the following lemma.

▶ **Lemma 7.** *Let $H = (V_H, E_H)$ be a graph, and let q be an integer satisfying $q \geq q(H)$. Consider a graph $G = (V, E)$ and a vertex cover $X \subseteq V$ of G of size k. Define $G' = (V', E')$ as the graph obtained from $G[X]$ by adding, for every non-empty set $S \subseteq X$ of size at most q such that $S \subseteq N_G(v)$ for some $v \in V \setminus X$, a vertex v_S adjacent to the vertices of S. Then the following holds.*

1. *The set X forms a vertex cover of G'.*
2. *The number of vertices in G' is at most $k + \sum_{i=1}^{q} \binom{k}{i}$.*
3. *The graph G' can be encoded in $\binom{k}{2} + \sum_{i=1}^{q} \binom{k}{i}$ bits.*
4. *The graph G is H-colorable if and only if the graph G' is H-colorable.*

Proof. By the definition of the graph G', every edge of G' connects either two vertices of X or a vertex of X to a vertex v_S, hence X forms a vertex cover of G'. It also follows from the definition that the number of vertices in G' does not exceed $k + \sum_{i=1}^{q} \binom{k}{i}$. Additionally, G' can be represented by a binary string that expresses the adjacencies inside $G[X]$ as well as whether the vertex v_S is included in G' or not, for each non-empty set $S \subseteq X$ of size $|S| \leq q$. Therefore, the graph G' can be encoded in $\binom{k}{2} + \sum_{i=1}^{q} \binom{k}{i}$ bits.

It remains to prove the fourth item of the lemma, namely, that the graph G is H-colorable if and only if the graph G' is H-colorable. Suppose first that G is H-colorable, and let $f : V \to V_H$ be a homomorphism from G to H. Consider the function $f' : V' \to V_H$ defined as follows. First, for every vertex $u \in X$, let $f'(u) = f(u)$. Since every edge of $G'[X]$ is also an edge of $G[X]$, its endpoints are mapped by f' to adjacent vertices in H. Now, every vertex in $V' \setminus X$ has the form v_S for a set $S \subseteq X$ of size $|S| \leq q$, where there exists a vertex $v \in V \setminus X$ with $S \subseteq N_G(v)$. We define $f'(v_S) = f(v)$ for such a vertex v. Since f and f' agree on X, it follows that $f'(v_S)$ is adjacent in H to $f'(u)$ for all vertices $u \in S = N_{G'}(v_S)$. We thus obtain that f' is a homomorphism from G' to H, so G' is H-colorable.

For the converse direction, suppose that G' is H-colorable, and let $f' : V' \to V_H$ be a homomorphism from G' to H. Consider the function $f : V \to V_H$ defined as follows. First, for every vertex $u \in X$, let $f(u) = f'(u)$. Since every edge of $G[X]$ is also an edge of

$G'[X]$, its endpoints are mapped by f to adjacent vertices in H. Next, consider any vertex $u \in V \setminus X$. Since X is a vertex cover of G, all the neighbors of u lie in X. We claim that there exists a vertex in H that is adjacent to all vertices $f(v)$ with $v \in N_G(u)$. To see this, let $T = \{f(v) \mid v \in N_G(u)\}$, and suppose for contradiction that the vertices of T have no common neighbor in H. By the definition of the non-adjacency witness number, there exists a non-empty set $T' \subseteq T$ of size $|T'| \leq q(H) \leq q$ with no common neighbor in H. Notice that there also exists a non-empty set $S \subseteq N_G(u)$ of size $|S| \leq q$ such that $T' = \{f(v) \mid v \in S\}$. However, the graph G' includes the corresponding vertex v_S, which is mapped by f' to a vertex in H that is adjacent to all vertices of T', a contradiction. This shows that the set T must admit a common neighbor in H, allowing us to define $f(u)$ as such a neighbor. It follows that f is a homomorphism from G to H, so G is H-colorable, as desired. ◀

We are ready to derive Theorem 1, which states that for every graph H, the H-Coloring problem parameterized by the vertex cover number k admits a kernel with $O(k^{q(H)})$ vertices and bit-size $O(k^{q(H)})$. Consequences for concrete graph classes appear in the full version of the paper.

Proof of Theorem 1. Let $H = (V_H, E_H)$ be a fixed graph, and set $q = q(H)$. If H is edgeless, then the H-Coloring problem is equivalent to determining whether an input graph is edgeless, and is thus solvable in polynomial time. Therefore, we may and will assume that H has at least one edge, which easily implies that $q \geq 2$.

The input of the H-Coloring problem parameterized by the vertex cover number k consists of a graph $G = (V, E)$ and a vertex cover $X \subseteq V$ of G of size $|X| = k$. Consider the kernelization algorithm that, given such an input, produces the graph $G' = (V', E')$ defined as in Lemma 7, that is, the graph obtained from $G[X]$ by adding, for every non-empty set $S \subseteq X$ of size $|S| \leq q$ such that $S \subseteq N_G(v)$ for some $v \in V \setminus X$, a vertex v_S adjacent to the vertices of S. Then, the algorithm returns the graph G' along with the set X.

The analysis of the algorithm is based on Lemma 7. By Item 1 of the lemma, the set X forms a vertex cover of G', so the pair (G', X) produced by the algorithm is a valid output. By Item 2, the number of vertices in G' satisfies $|V'| \leq k + \sum_{i=1}^{q} \binom{k}{i} \leq O(k^q)$, and by Item 3, the number of bits needed to encode G' is at most $\binom{k}{2} + \sum_{i=1}^{q} \binom{k}{i}$, which is $O(k^q)$ since $q \geq 2$. Item 4 ensures the correctness of the kernelization algorithm. Finally, since q is a fixed constant, one can enumerate all subsets of X of size at most q and construct the graph G' in polynomial time. The proof is now complete. ◀

3.2 The Algebraic Kernel

Our algebraic kernel for the H-Coloring problem parameterized by the vertex cover number uses the notion of independent representations, a generalization of orthogonal representations of graphs.

3.2.1 Orthogonal and Independent Graph Representations

We begin with formal definitions of these two representation types.

▶ **Definition 8.** *A* d-dimensional orthogonal representation *of a graph* $G = (V, E)$ *over a field* $\mathbb{F}$ *is an assignment of a vector* $x_v \in \mathbb{F}^d$ *with* $\langle x_v, x_v \rangle \neq 0$ *to each vertex* $v \in V$*, such that for all vertices* $u, v \in V$*, if* $\{u, v\} \in E$ *then* $\langle x_u, x_v \rangle = 0$*. The orthogonal representation is called* faithful *if for all* $u, v \in V$*, it holds that* $\{u, v\} \in E$ *if and only if* $\langle x_u, x_v \rangle = 0$*.*

▶ **Definition 9.** *A* d-dimensional independent representation *of a graph* $G = (V, E)$ *over a field* $\mathbb{F}$ *is an assignment of a vector* $x_v \in \mathbb{F}^d$ *to each vertex* $v \in V$*, such that for every vertex* $v \in V$*, it holds that* $x_v \notin \operatorname{span}(\{x_w \mid w \in N_G(v)\})$*. The independent representation is called* faithful *if for all* $u, v \in V$*, it holds that* $\{u, v\} \in E$ *if and only if* $x_u \in \operatorname{span}(\{x_w \mid w \in N_G(v)\})$*. Notice that the forward implication holds trivially.*

We note that all vectors in an independent representation of a graph are nonzero. It can be verified that a graph has a 2-dimensional independent representation over a field if and only if it is bipartite.

▶ Remark 10. Every (faithful) orthogonal representation of a graph $G = (V, E)$ over a field $\mathbb{F}$ is also a (faithful) independent representation of G over $\mathbb{F}$. To see this, consider an orthogonal representation $(x_v)_{v \in V}$ of G over $\mathbb{F}$. For each vertex $v \in V$, the vector x_v is orthogonal to the vectors x_w with $w \in N_G(v)$, and thus to the subspace they span, but not to itself, so $x_v \notin \operatorname{span}(\{x_w \mid w \in N_G(v)\})$. This shows that $(x_v)_{v \in V}$ forms an independent representation of G. If the orthogonal representation $(x_v)_{v \in V}$ of G is faithful, then for all vertices $u, v \in V$ with $\{u, v\} \notin E$, the vector x_v is not orthogonal to x_u but is orthogonal to the vectors x_w with $w \in N_G(v)$, implying that $x_u \notin \operatorname{span}(\{x_w \mid w \in N_G(v)\})$. This confirms that $(x_v)_{v \in V}$ forms a faithful independent representation of G in this case.

We proceed by presenting several lemmas that will be used in the analysis of the algebraic kernel. Their proofs are deferred to the full version of the paper. The first lemma shows that the dimension of any faithful independent representation of a graph forms an upper bound on its non-adjacency witness number (recall Definition 6).

▶ **Lemma 11.** *For a graph* G*, a positive integer* d*, and a field* $\mathbb{F}$*, if* G *has a faithful* d*-dimensional independent representation over* $\mathbb{F}$*, then* $q(G) \leq d$*.*

The following lemma states that any faithful independent representation over a field can be transformed into one of the same dimension in which every vector starts with 1, over any sufficiently large extension field.

▶ **Lemma 12.** *Let* $G = (V, E)$ *be a graph, let* d *be a positive integer, and let* $\mathbb{F} \subseteq \mathbb{K}$ *be fields such that* $\mathbb{K}$ *is an extension field of* $\mathbb{F}$ *with* $|\mathbb{K}| > |V|$*. If* G *has a faithful* d*-dimensional independent representation over* $\mathbb{F}$*, then it also has a faithful* d*-dimensional independent representation over* $\mathbb{K}$*, in which all vectors have* 1 *as their first entry.*

We finally observe that every graph admits a faithful independent representation of dimension one greater than its maximum degree over any sufficiently large field.

▶ **Lemma 13.** *Let* $G = (V, E)$ *be a graph, let* $\mathbb{F}$ *be a field with* $|\mathbb{F}| \geq |V|$*, and set* $d = \Delta(G)+1$*. Then* G *has a faithful* d*-dimensional independent representation over* $\mathbb{F}$*.*

3.2.2 The Algorithm

We are ready to present our algebraic kernel and confirm Theorem 2. To state the result in full generality, we borrow the following terminology from [25]. A field is said to be *efficient* if field operations and Gaussian elimination can be performed in polynomial time in the size of a reasonable input encoding. Note that all finite fields and the real field $\mathbb{R}$ (restricted to rational numbers) are efficient.

▶ **Theorem 14.** *For a graph* H*, a positive integer* d*, and an efficient field* $\mathbb{F}$*, suppose that there exists a faithful* d*-dimensional independent representation of* H *over* $\mathbb{F}$*. Then the* H-Coloring *problem parameterized by the vertex cover number* k *admits a kernel with* $O(k^{d-1})$ *vertices and bit-size* $O(k^{d-1} \cdot \log k)$*.*

Proof. Consider a graph $H = (V_H, E_H)$, a positive integer d, and an efficient field $\mathbb{F}$, such that H has a faithful d-dimensional independent representation over $\mathbb{F}$. We may assume that $d \geq 3$, as otherwise H is bipartite, and the H-Coloring problem can be solved in polynomial time. We define a field $\mathbb{K}$ as follows. If $\mathbb{F}$ is finite, then we choose $\mathbb{K}$ to be some finite extension field of $\mathbb{F}$ whose order exceeds $|V_H|$ (e.g., a finite field of order $|\mathbb{F}|^\ell$ for the smallest positive integer ℓ satisfying $|\mathbb{F}|^\ell > |V_H|$). If $\mathbb{F}$ is infinite, then $\mathbb{K}$ is simply taken to be $\mathbb{F}$. In both cases, the field $\mathbb{K}$ is efficient and satisfies $|\mathbb{K}| > |V_H|$.

The input of the H-Coloring problem parameterized by the vertex cover number k consists of a graph $G = (V, E)$ and a vertex cover $X \subseteq V$ of G of size $|X| = k$. Consider the kernelization algorithm that, given such an input, performs the following steps.

1. Produce the graph $G' = (V', E')$ defined as in Lemma 7 for $q = d$. Namely, G' is the graph obtained from $G[X]$ by adding, for every non-empty set $S \subseteq X$ of size $|S| \leq d$ such that $S \subseteq N_G(v)$ for some $v \in V \setminus X$, a vertex v_S adjacent to the vertices of S.
2. Associate with each vertex $u \in X$ a d-dimensional symbolic vector y_u over the field $\mathbb{K}$, where the first entry of y_u is fixed to 1 and the remaining $d-1$ entries are free variables. Note that the total number of variables is $k \cdot (d-1)$.
3. Consider the collection Y of all sets $S \subseteq X$ of size precisely d for which the vertex v_S is included in the graph G', i.e., $Y = \{S \subseteq X \mid v_S \in V',\ |S| = d\}$. For each such set $S \in Y$, define a polynomial p_S over the variables of $(y_u)_{u \in X}$ as the determinant over the field $\mathbb{K}$ of a $d \times d$ matrix whose columns are the vectors y_u with $u \in S$ (ordered arbitrarily).
4. Consider the subspace of polynomials $W = \text{span}(\{p_S \mid S \in Y\})$, and compute a set $Y' \subseteq Y$ such that the polynomials in $\{p_S \mid S \in Y'\}$ form a basis of W.
5. Let $G'' = (V'', E'')$ be the graph obtained from G' by removing all vertices v_S with $S \in Y \setminus Y'$, and return it along with the set X.

Let us emphasize that the given independent representation of H is not used by the algorithm itself, but it plays a crucial role in its correctness proof. Yet, the algorithm uses the field $\mathbb{K}$, over which it computes the polynomials p_S for $S \in Y$ and a basis of the subspace W.

We now analyze the algorithm and show that it satisfies the assertion of the theorem. First, by Item 1 of Lemma 7, the set X forms a vertex cover of G', and so also forms a vertex cover of its subgraph G''. Therefore, the pair (G'', X) produced by the algorithm is a valid output. The vertex set of G'' consists of the k vertices of X, vertices v_S with $|S| \leq d-1$, whose number is at most $\sum_{i=1}^{d-1} \binom{k}{i}$, and the vertices v_S with $S \in Y'$. To bound the size of Y', we observe that each polynomial p_S with $S \in Y$ is multilinear and homogeneous of degree $d-1$. Indeed, the determinant polynomial of a $d \times d$ matrix is a linear combination of monomials of degree d, where each monomial is a product of exactly d entries, one from each row. Since p_S is defined as the determinant of a $d \times d$ matrix, whose first-row entries are all ones and all other entries are distinct variables, it follows that p_S has the required form. This implies that W is a subspace of the space of multilinear homogeneous polynomials of degree $d-1$ in $k \cdot (d-1)$ variables, whose dimension is $\binom{k \cdot (d-1)}{d-1}$. Since the size of a basis of such a subspace cannot exceed the dimension of the ambient space, it follows that $|Y'| \leq \binom{k \cdot (d-1)}{d-1}$. We obtain that the number of vertices in G'' satisfies

$$|V''| \leq k + \sum_{i=1}^{d-1} \binom{k}{i} + \binom{k \cdot (d-1)}{d-1} \leq O(k^{d-1}),$$

where the last inequality holds because d is a fixed constant. The graph G'' can be expressed by a binary string that represents the adjacencies in $G''[X]$ and the sets $S \subseteq X$ with $|S| \leq d$ for which the vertex v_S is included in G''. Since each such set S can be encoded in $d \cdot \lceil \log k \rceil$ bits, the total number of bits needed to encode G'' does not exceed $\binom{k}{2} + |V'' \setminus X| \cdot d \cdot \lceil \log k \rceil$,

which is bounded by $O(k^{d-1} \cdot \log k)$, given that $d \geq 3$. We further notice that G'' can be constructed in polynomial time. Indeed, the construction of G' involves enumerating all subsets of X of size at most d. Then, to determine the vertices retained in G'', we compute the set Y' by applying Gaussian elimination over the efficient field $\mathbb{K}$ on a system with $\binom{k \cdot (d-1)}{d-1}$ variables, which can be performed in polynomial time.

We turn to proving the correctness of the algorithm, namely, that G is H-colorable if and only if G'' is H-colorable. By assumption, the graph H has a faithful d-dimensional independent representation over $\mathbb{F}$. Since $\mathbb{K}$ is an extension field of $\mathbb{F}$ with $|\mathbb{K}| > |V_H|$, Lemma 12 implies that H also has a faithful d-dimensional independent representation $(x_v)_{v \in V_H}$ over $\mathbb{K}$, in which all vectors have 1 as their first entry. By Lemma 11, we have $q(H) \leq d$, which allows us to apply Item 4 of Lemma 7 and conclude that the graph G is H-colorable if and only if the graph G' is H-colorable. It is thus sufficient to prove that G' is H-colorable if and only if G'' is H-colorable. Since G'' is a subgraph of G', it is clear that if G' is H-colorable then so is G''. It remains to show that if G'' is H-colorable, then so is G'.

Suppose that the graph G'' is H-colorable, and let $f'' : V'' \to V_H$ be a homomorphism from G'' to H. We define an assignment ρ of elements from $\mathbb{K}$ to the variables of $(y_u)_{u \in X}$ by assigning to each variable vector y_u the vector $x_{f''(u)}$ from the given independent representation over $\mathbb{K}$. Note that this is possible because the first entry in y_u is 1, just as in $x_{f''(u)}$, while the other entries in y_u are free variables. We claim that for every set $S \in Y$, the polynomial p_S defined in Item 3 of the algorithm vanishes at the assignment ρ. It suffices to verify this only for sets $S \in Y'$, since the set $\{p_S \mid S \in Y'\}$ forms a basis of the subspace $W = \text{span}(\{p_S \mid S \in Y\})$. Consider a set $S \in Y'$, and recall that the vertex v_S is adjacent in G'' to the vertices of S, hence $f''(v_S)$ is adjacent in H to all vertices $f''(u)$ with $u \in S$. By the definition of an independent representation, the vector $x_{f''(v_S)}$ does not lie in the linear subspace spanned by the vectors $x_{f''(u)}$ with $u \in S$. In particular, the dimension of this subspace is strictly smaller than d, so the determinant of a $d \times d$ matrix whose columns are the vectors $x_{f''(u)}$ with $u \in S$ is zero. This implies that p_S evaluates to zero at the assignment ρ, as desired.

We now show that there exists a homomorphism $f' : V' \to V_H$ from G' to H. We start by defining $f'(u) = f''(u)$ for each $u \in X$. Since every edge of $G'[X]$ is also an edge of $G''[X]$, its endpoints are mapped by f' to adjacent vertices in H. Next, for every vertex v_S in G' with $|S| \leq d-1$, set $f'(v_S) = f''(v_S)$. Since the neighborhood of such a vertex v_S is contained in X and is identical to its neighborhood in G'', the vertex $f'(v_S)$ is adjacent in H to all the images under f' of the neighbors of v_S in G'. It remains to show that for every set $S \in Y$, the vertex v_S in G' can be assigned a vertex in H that is adjacent to all vertices $f'(u)$ with $u \in S$.

To see this, consider a set $S \in Y$, and recall that $|S| = d$ and that the polynomial p_S vanishes at the assignment ρ described above. Therefore, the vectors of $\{x_{f'(w)} \mid w \in S\}$ span a subspace of dimension smaller than d, so there exists a non-empty set $S' \subseteq S$ of size $|S'| \leq d-1$ such that the vectors of $\{x_{f'(w)} \mid w \in S'\}$ span the exact same subspace. Since the vertex v_S is included in G', it follows that $S \subseteq N_G(v)$ for some $v \in V \setminus X$, so by $S' \subseteq S$, the vertex $v_{S'}$ is included in G' as well as in G''. We define $f'(v_S) = f''(v_{S'})$, and show that $f''(v_{S'})$ is adjacent in H to all vertices $f'(u)$ with $u \in S$. Indeed, for every vertex $u \in S$, it holds that

$$\begin{aligned} x_{f'(u)} \in \text{span}(\{x_{f'(w)} \mid w \in S\}) &= \text{span}(\{x_{f'(w)} \mid w \in S'\}) \\ &\subseteq \text{span}(\{x_v \mid v \in N_H(f''(v_{S'}))\}), \end{aligned} \tag{1}$$

where the containment holds because the vertices of S' are adjacent in G'' to $v_{S'}$, so they are mapped by f'', and thus by f', to neighbors of $f''(v_{S'})$ in H. By the faithfulness of the given independent representation of H, it follows from (1) that the vertices $f'(u)$ and $f''(v_{S'})$ are

adjacent in H. We conclude that $f''(v_{S'})$ is adjacent in H to all vertices $f'(u)$ with $u \in S$, as desired. This gives us a homomorphism from G' to H, implying that G' is H-colorable, as required. ◀

We end this section with the observation that Theorem 14 strengthens the kernelization bounds implied by the prior work [24]. Indeed, by Lemma 13, every graph H has a faithful independent representation of dimension $\Delta(H)+1$ over, for instance, the real field $\mathbb{R}$. Invoking Theorem 14 yields a kernel for the H-Coloring problem parameterized by the vertex cover number k with $O(k^{\Delta(H)})$ vertices and bit-size $O(k^{\Delta(H)} \cdot \log k)$. As illustrated in the full version of the paper, the kernel obtained via Theorem 14 is sometimes markedly more compact.

4 Concluding Remarks

This paper studies the kernelization complexity of H-Coloring problems parameterized by the vertex cover number and presents two kernels for them. The first is purely combinatorial, with size governed by the non-adjacency witness number of H, while the second is more algebraic in nature, relying on the existence of a low-dimensional faithful independent representation of H over some field. We have shown that the combinatorial kernel, despite its simplicity, achieves significant improvements over prior work for certain graphs H, and that the algebraic kernel can sometimes save a further linear factor in the kernel size. Moreover, we have provided lower bounds on the kernel size for projective graphs H and have demonstrated that in certain cases our kernels attain near-optimal bounds.

The natural challenge raised by this work is to determine, for general graphs H, the exact polynomial behavior of the kernel complexity of the H-Coloring problem parameterized by the vertex cover number. In particular, it would be valuable to obtain lower bounds on the kernel complexity of the problem for non-bipartite core graphs H that are not projective. For such graphs H, it would also be interesting to decide whether the H-Coloring problem admits no sub-quadratic compression when parameterized by the number of vertices. These are the remaining unresolved instances of a question raised in [3]. Finally, it would be of interest to explore the kernel complexity of H-Coloring problems under alternative parameterizations, a direction that has already seen several developments, e.g., in [23, 24, 18].

References

1 Ziv Bar-Yossef, Yitzhak Birk, T. S. Jayram, and Tomer Kol. Index coding with side information. *IEEE Trans. Inform. Theory*, 57(3):1479–1494, 2011. Preliminary version in FOCS'06. doi:10.1109/TIT.2010.2103753.

2 Peter J. Cameron, Ashley Montanaro, Michael W. Newman, Simone Severini, and Andreas J. Winter. On the quantum chromatic number of a graph. *Electron. J. Comb.*, 14(1):R81, 2007. doi:10.37236/999.

3 Hubie Chen, Bart M. P. Jansen, Karolina Okrasa, Astrid Pieterse, and Pawel Rzążewski. Sparsification lower bounds for list H-coloring. *ACM Trans. Comput. Theory*, 15(3–4):8:1–23, 2023. Preliminary version in ISAAC'20. doi:10.1145/3612938.

4 Bruno Codenotti, Pavel Pudlák, and Giovanni Resta. Some structural properties of low-rank matrices related to computational complexity. *Theor. Comput. Sci.*, 235(1):89–107, 2000. doi:10.1016/S0304-3975(99)00185-1.

5 Marek Cygan, Fedor V. Fomin, Alexander Golovnev, Alexander S. Kulikov, Ivan Mihajlin, Jakub Pachocki, and Arkadiusz Socala. Tight lower bounds on graph embedding problems. *J. ACM*, 64(3):18:1–22, 2017. Preliminary versions in ICALP'15 and SODA'16. doi:10.1145/3051094.

6 Mitre Costa Dourado, Fábio Protti, and Jayme Luiz Szwarcfiter. Computational aspects of the Helly property: A survey. *J. Braz. Comput. Soc.*, 12(1):7–33, 2006. doi:10.1007/BF03192385.

7 Tomás Feder, Pavol Hell, and Jing Huang. List homomorphisms and circular arc graphs. *Combinatorica*, 19(4):487–505, 1999. doi:10.1007/S004939970003.

8 Fedor V. Fomin, Pinar Heggernes, and Dieter Kratsch. Exact algorithms for graph homomorphisms. *Theory Comput. Syst.*, 41(2):381–393, 2007. Preliminary version in FCT'05. doi:10.1007/S00224-007-2007-X.

9 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: Theory of Parameterized Preprocessing*. Cambridge University Press, 2019.

10 Robert Ganian, Thekla Hamm, Viktoriia Korchemna, Karolina Okrasa, and Kirill Simonov. The fine-grained complexity of graph homomorphism parameterized by clique-width. *ACM Trans. Algorithms*, 20(3):19, 2024. Preliminary version in ICALP'22. doi:10.1145/3652514.

11 Alexander Golovnev and Ishay Haviv. The (generalized) orthogonality dimension of (generalized) Kneser graphs: Bounds and applications. *Theory of Computing*, 18(22):1–22, 2022. Preliminary version in CCC'21. doi:10.4086/TOC.2022.V018A022.

12 Alexander Golovnev, Oded Regev, and Omri Weinstein. The minrank of random graphs. *IEEE Trans. Inform. Theory*, 64(11):6990–6995, 2018. Preliminary version in RANDOM'17. doi:10.1109/TIT.2018.2810384.

13 Marina Groshaus, Min Chih Lin, and Jayme Luiz Szwarcfiter. On neighborhood–Helly graphs. *Discret. Appl. Math.*, 216(1):191–202, 2017. doi:10.1016/J.DAM.2016.04.029.

14 Marina Groshaus and Jayme Luiz Szwarcfiter. Biclique–Helly graphs. *Graphs Comb.*, 23(6):633–645, 2007. doi:10.1007/S00373-007-0756-6.

15 Willem H. Haemers. On some problems of Lovász concerning the Shannon capacity of a graph. *IEEE Trans. Inform. Theory*, 25(2):231–232, 1979. doi:10.1109/TIT.1979.1056027.

16 Ishay Haviv. On minrank and the Lovász theta function. In *International Conference on Approximation Algorithms for Combinatorial Optimization Problems (APPROX'18)*, pages 13:1–15, 2018. doi:10.4230/LIPICS.APPROX-RANDOM.2018.13.

17 Ishay Haviv. On minrank and forbidden subgraphs. *ACM Transactions on Computation Theory (TOCT)*, 11(4):20:1–13, 2019. Preliminary version in RANDOM'18. doi:10.1145/3322817.

18 Ishay Haviv and Dror Rabinovich. Kernelization for orthogonality dimension. In *Proc. of the 19th International Symposium on Parameterized and Exact Computation (IPEC'24)*, pages 8:1–17, 2024. doi:10.4230/LIPICS.IPEC.2024.8.

19 Ishay Haviv and Dror Rabinovich. A near-optimal kernel for a coloring problem. *Discret. Appl. Math.*, 377:66–73, 2025. doi:10.1016/J.DAM.2025.06.065.

20 Pavol Hell and Jaroslav Nešetřil. On the complexity of H-coloring. *J. Comb. Theory, Ser. B*, 48(1):92–110, 1990. doi:10.1016/0095-8956(90)90132-J.

21 Pavol Hell and Jaroslav Nešetřil. *Graphs and homomorphisms*, volume 28 of *Oxford lecture series in mathematics and its applications*. Oxford University Press, 2004.

22 Bart M. P. Jansen. The Power of Data Reduction: Kernels for Fundamental Graph Problems. Ph.D. Thesis, Utrecht University, 2013.

23 Bart M. P. Jansen and Stefan Kratsch. Data reduction for graph coloring problems. *Inf. Comput.*, 231:70–88, 2013. Preliminary version in FCT'11. doi:10.1016/J.IC.2013.08.005.

24 Bart M. P. Jansen and Astrid Pieterse. Optimal data reduction for graph coloring using low-degree polynomials. *Algorithmica*, 81(10):3865–3889, 2019. Preliminary version in IPEC'17. doi:10.1007/S00453-019-00578-5.

25 Bart M. P. Jansen and Astrid Pieterse. Optimal sparsification for some binary CSPs using low-degree polynomials. *ACM Trans. Comput. Theory*, 11(4):28:1–26, 2019. Preliminary version in MFCS'16. doi:10.1145/3349618.

26 Benoît Larose. Families of strongly projective graphs. *Discuss. Math. Graph Theory*, 22(2):271–292, 2002. doi:10.7151/DMGT.1175.

27 László Lovász. Kneser's conjecture, chromatic number, and homotopy. *J. Comb. Theory, Ser. A*, 25(3):319–324, 1978. doi:10.1016/0097-3165(78)90022-5.

28 László Lovász. On the Shannon capacity of a graph. *IEEE Trans. Inform. Theory*, 25(1):1–7, 1979. doi:10.1109/TIT.1979.1055985.

29 László Lovász. *Graphs and Geometry*, volume 65. Colloquium Publications, 2019.

30 László Lovász, Michael Saks, and Alexander Schrijver. Orthogonal representations and connectivity of graphs. *Linear Algebra Appl.*, 114–115:439–454, 1989.

31 Hiroshi Maehara and Vojtech Rödl. On the dimension to represent a graph by a unit distance graph. *Graphs Comb.*, 6(4):365–367, 1990. doi:10.1007/BF01787703.

32 Hermann A. Maurer, Ivan Hal Sudborough, and Emo Welzl. On the complexity of the general coloring problem. *Inf. Control.*, 51(2):128–145, 1981. doi:10.1016/S0019-9958(81)90226-6.

33 Arya Mazumdar. Storage capacity of repairable networks. *IEEE Trans. Inform. Theory*, 61(11):5810–5821, 2015. Preliminary versions in ISIT'14 and Allerton'14. doi:10.1109/TIT.2015.2472521.

34 Jaroslav Nešetřil. Representations of graphs by means of products and their complexity. In *Proc. of the Mathematical Foundations of Computer Science (MFCS'81)*, pages 94–102, 1981.

35 Karolina Okrasa and Pawel Rzążewski. Fine-grained complexity of the graph homomorphism problem for bounded-treewidth graphs. *SIAM J. Comput.*, 50(2):487–508, 2021. Preliminary version in SODA'20. doi:10.1137/20M1320146.

36 René Peeters. Orthogonal representations over finite fields and the chromatic number of graphs. *Combinatorica*, 16(3):417–431, 1996. doi:10.1007/BF01261326.

37 Marta Piecyk and Pawel Rzążewski. Fine-grained complexity of the list homomorphism problem: Feedback vertex set and cutwidth. In *Proc. of the 38th International Symposium on Theoretical Aspects of Computer Science (STACS'21)*, pages 56:1–17, 2021.

38 Søren Riis. Information flows, graphs and their guessing numbers. *Electron. J. Comb.*, 14(1):R44, 2007. Preliminary version in NETCOD'06. doi:10.37236/962.

39 Pawel Rzążewski. Exact algorithm for graph homomorphism and locally injective graph homomorphism. *Inf. Process. Lett.*, 114(7):387–391, 2014. doi:10.1016/J.IPL.2014.02.012.

40 Karthikeyan Shanmugam, Alexandros G. Dimakis, and Michael Langberg. Local graph coloring and index coding. In *Proc. of the IEEE International Symposium on Information Theory (ISIT'13)*, pages 1152–1156, 2013. doi:10.1109/ISIT.2013.6620407.

41 Magnus Wahlström. New plain-exponential time classes for graph homomorphism. *Theory Comput. Syst.*, 49(2):273–282, 2011. Preliminary version in CSR'09. doi:10.1007/S00224-010-9261-Z.

42 Chee-Keng Yap. Some consequences of non-uniform conditions on uniform classes. *Theor. Comput. Sci.*, 26(3):287–300, 1983. doi:10.1016/0304-3975(83)90020-8.

Boundaried Kernelization via Representative Sets

Leonid Antipov
Humboldt-Universität zu Berlin, Germany

Stefan Kratsch
Humboldt-Universität zu Berlin, Germany

Abstract

A kernelization is an efficient algorithm that given an instance of a parameterized problem returns an equivalent instance of size bounded by some function of the input parameter value. It is quite well understood which problems do or (conditionally) do not admit a kernelization where this size bound is polynomial, a so-called polynomial kernelization. Unfortunately, such polynomial kernelizations are known only in fairly restrictive settings where a small parameter value corresponds to a strong restriction on the global structure on the instance. Motivated by this, Antipov and Kratsch [WG 2025] proposed a local variant of kernelization, called boundaried kernelization, that requires only local structure to achieve a local improvement of the instance, which is in the spirit of protrusion replacement used in meta-kernelization [Bodlaender et al. JACM 2016]. They obtain polynomial boundaried kernelizations as well as (unconditional) lower bounds for several well-studied problems in kernelization.

In this work, we leverage the matroid-based techniques of Kratsch and Wahlström [JACM 2020] to obtain randomized polynomial boundaried kernelizations for s-MULTIWAY CUT, DELETABLE TERMINAL MULTIWAY CUT, ODD CYCLE TRANSVERSAL, and VERTEX COVER[oct], for which randomized polynomial kernelizations in the usual sense were known before. A priori, these techniques rely on the global connectivity of the graph to identify reducible (irrelevant) vertices. Nevertheless, the separation of the local part by its boundary turns out to be sufficient for a local application of these methods.

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

Keywords and phrases Parameterized complexity, boundaried kernelization, local preprocessing, representative sets method

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.6

Related Version *Full Version*: `https://arxiv.org/abs/2510.00832` [2]

Funding *Leonid Antipov*: Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project number 526215872.

1 Introduction

(Polynomial) kernelization is a very successful notion for rigorously studying efficient preprocessing for hard problems. A kernelization is an efficient algorithm that given an instance of a parameterized problem returns an equivalent instance of size bounded by some function of the input parameter value. It is a polynomial kernelization if this function is polynomially bounded. By now, it is quite well understood which problems do or (conditionally) do not admit a polynomial kernelization. In this way, we have learned what structural properties are helpful for a provable size reduction through efficient preprocessing, which is unlikely in general unless $\mathsf{P} = \mathsf{NP}$. Unfortunately, most polynomial kernelizations are for parameters that take low values only on instances with very restricted *global structure*, e.g., only parameter many vertex deletions away from a known tractable special case of the problem. This is an issue because the size guarantee of a kernelization is only nontrivial when the parameter is small, otherwise we could just leave the instance as is.

© Leonid Antipov and Stefan Kratsch;
licensed under Creative Commons License CC-BY 4.0

Motivated by this, Antipov and Kratsch [1] proposed a local variant of kernelization, called *boundaried kernelization*, inspired by protrusion replacement in meta-kernelization (Bodlaender et al. [3]). Roughly, this expects as input a boundaried graph G_B and will return a boundaried graph G'_B of size some function of the chosen parameter plus the boundary size (and possibly a shift Δ in solution value). The two graphs are equivalent in the sense that gluing any boundaried graph H_B will result in equivalent instances (up to shift of Δ for optimization). Intuitively, such a preprocessing only needs a local part of a large graph to have beneficial structure and sufficiently small connection to the rest of the input, for the size bound to imply a size reduction. In this sense, boundaried kernelization relaxes the required structural conditions (from global to local), though of course also the size bound is only local (namely only for the boundaried local part). Since we can always run a boundaried kernelization with empty boundary, however, it is by itself in fact a strengthening of kernelization (modulo some technical aspects due to the variety of different parameterizations). Thus, it is natural and interesting to ask, which known polynomial kernelizations can be strengthened to polynomial boundaried kernelizations.

In [1], polynomial boundaried kernelizations were obtained for VERTEX COVER[vc], VERTEX COVER[fvs], FEEDBACK VERTEX SET[fvs], LONG CYCLE[vc], LONG PATH[vc], HAMILTONIAN CYCLE[vc], and HAMILTONIAN PATH[vc].[1] In contrast, CLUSTER EDITING[cvd], CLUSTER EDITING[ce], TREE DELETION SET[vc], TREE DELETION SET[tds], and DOMINATING SET[vc] were shown to (unconditionally) not admit a polynomial boundaried kernelization.[2] Existence of unconditional lower bounds is not surprising and relies on exhibiting an unbounded (or at least too large) family of non-equivalent boundaried graphs. That being said, it is somewhat surprising that simple local reduction rules can sometimes be leveraged also for boundaried kernelization, while they otherwise fail completely.

There are of course plenty of other graph problems with polynomial kernelizations to explore. Next to well-studied problems (as above) and cases with somewhat special kernelization (as TREE DELETION SET[tds]), it would be interesting to know how inherently global tools fare in the boundaried setting. Here, the matroid-based techniques of Kratsch and Wahlström [13] come to mind, as they rely on the properties of certain matroids defined on the entire graph, while also enabling the first (and so far only) polynomial kernelizations for certain cut and feedback problems. Another interesting one would be the randomized polynomial compression for STEINER CYCLE[$|T|$] due to Wahlström [18] but likely one would first have to strengthen it to a kernelization, as it outputs a somewhat contrived matrix problem not known to be in NP.

Our work. Motivated by the question of whether the global matroid-based techniques of Kratsch and Wahlström [13] can also be adapted for local preprocessing, we study the same problems in the boundaried setting. We are able to strengthen several results to polynomial boundaried kernelizations.

▶ **Theorem 1.** *The following parameterized problems admit randomized polynomial boundaried kernelization:* s-MULTIWAY CUT[local solution] *(with a minor restriction described below),* DELETABLE TERMINAL MULTIWAY CUT[local solution]*,* ODD CYCLE TRANSVERSAL[local solution]*, and* VERTEX COVER[oct].

[1] Parameterized problems are denoted as problem name followed by parameter in brackets: These include parameterization by the size of a given vertex cover (vc), feedback vertex set (fvs), cluster vertex deletion set (cvd), cluster editing set (ce), and tree deletion set (tds), respectively.

[2] It was known that DOMINATING SET[vc] has no polynomial kernelization unless NP $\subseteq$ coNP/poly, unlike all other problems listed here, but exclusion of polynomial boundaried kernelization is unconditional.

The main idea lies in the power of the gammoids and the resulting cut covers Z which contain the vertices of an exponential number of min-cut queries. While Kratsch and Wahlström applied this on gammoids with sources being the terminal set, or an approximate odd cycle transversal, we additionally put in the boundary vertices as sources. As a result, this family of min-cut queries whose optimal answers are all covered by Z, contain in particular all possible combinations needed in order to complete an optimal global solution for any forced behavior of the boundary vertices in the solution. For instance, in s-MULTIWAY CUT, where the solution is a vertex set X such that each of the s given terminals $t \in T$ is contained in its own component in $G - X$, we are able to preserve an optimal solution for every choice of putting any boundary vertex either into the solution, the component of some terminal, or some unrelated component. Or in ODD CYCLE TRANSVERSAL, where $G - X$ has to be a bipartite graph, the set Z contains the local part of an optimal solution for every choice of putting B vertices in the solution or one of the two parts of the bipartite graph $G - X$.

Note however for the problem s-MULTIWAY CUT, that, for G_B being the local graph, T_G a set of terminals among G-vertices, and H_B, T_H the rest of the global graph with the remaining terminals among the H-vertices, if we allow T_H to also contain B-vertices, this forces us to be able to adapt to any pair of the boundary vertices to be terminals, and thus we need to store at least some information about the disjoint path sets between any such pair. Since the sizes of these path sets are not bounded, this means that we need to be able to store an unbounded amount of information, which directly rules out any effective preprocessing.

▶ **Theorem 2.** *In general, the parameterized problem* s-MULTIWAY CUT[local solution] *does not admit any boundaried kernelization.*

So, the slightly restricted case that we are referring to in Theorem 1 is that we forbid the terminal-part T_H that is unknown for the kernelization, to contain any boundary vertices. This restriction is enough in order to obtain a boundaried kernelization, and it fits with the concept of local kernelization because the terminals in B are known in that setting. (That being said, in general, it is appealing to push to the most general setting for getting a polynomial boundaried kernelization. This includes having the preprocessing be fully independent of possible glued graphs H_B.)

Related work. Most closely related is the already mentioned work on meta-kernelization via protrusion replacement (subgraphs with constant treewidth and constant boundary size). Inherently, the machinery for that does not lead to any polynomial size bounds, and that is not to be expected. Generally, the perspective of boundaried graphs is essential for dynamic programming on path and tree decompositions (cf. [4]). In that setting, there is no hope for polynomial boundaried kernelizations because it would generalize the (likely) infeasible case of problems parameterized by path/treewidth regarding polynomial kernelization. Instead, our settings have more restrictive structure than small treewidth (as is usual for polynomial kernelization) and we obtain polynomial size bounds.

Fomin et al. [6] use a different form of protrusion for (CONNECTED) DOMINATING SET$[k]$ where they require constant boundary size and a constant local solution size. It is used very differently than our (parameter) local cost, but is certainly reminiscent. Clearly, a small or constant local cost is a natural quality of interest amongst more explicit structural parameters like the size of modulators.

Work by Jansen and Kratsch [10] for preprocessing the feasibility problem of integer linear programs showed how to shrink subsystems with constant boundary size and either constant treewidth or total unimodularity to size polynomial in the domain. This predates the present notion of boundaried kernelization, like protrusion-replacement, but similarly does not yield size polynomial in the boundary size.

Organization. We give preliminaries regarding graph problems and boundaried kernelization in Section 2 and restate the tools from representative sets for matroids in Section 3. In Sections 4, 5, 6, and 7 we describe the randomized polynomial boundaried kernelizations for s-MULTIWAY CUT[local solution], DELETABLE TERMINAL MULTIWAY CUT[local solution], ODD CYCLE TRANSVERSAL[local solution], and VERTEX COVER[oct], respectively. Finally, we conclude in Section 8. Throughout the paper, we denote results by $\star$, if the corresponding proof is omitted and to be found in the full version [2].

2 Preliminaries

Graphs and graph problems. We use the usual notion of decision and optimization problems as well as standard graph theoretic notation mostly following Diestel [5]. Our graphs are finite, simple, undirected, loopless, and unweighted, unless explicitly stated otherwise. A *mixed* graph contains both directed and undirected edges. For a vertex set $V' \subseteq V(G)$, we denote by $N_{G,V'}(v)$ the set of neighbors of v that are contained in V', i.e., $N_{G,V'}(v) := N_G(v) \cap V'$. For vertex set $W \subseteq V(G)$, we extend this notion to $N_{G,V'}(W)$.

A *(vertex) annotated graph* is a tuple $(G, \mathcal{T})$, where G is an undirected graph and $\mathcal{T}$ itself is a tuple of subsets of $V(G)$. A *decision or optimization problem on (annotated) graphs* consists of instances encoding (annotated) graphs, respectively. A problem on graphs, i.e., without annotation $\mathcal{T}$, is also called a *pure graph problem*. We define the pure graph optimization problems VERTEX COVER and ODD CYCLE TRANSVERSAL in the usual way.

A set $S \subseteq V(G)$ is called a *(multiway) cut* for a graph G and a set $T = \{t_1, \ldots, t_k\} \subseteq V(G)$ of given *terminals* if in graph $G' = G - S$ for every pair $u, v \in T$ there is no path between u and v in $G - S$. More generally, if we are given graph G and a tuple of terminal sets $\mathcal{T} = (T_1, \ldots, T_k)$, then a cut for $(G, \mathcal{T})$ is a set $S \subseteq V(G)$ such that for any $i \neq j \in [k]$ and any $u \in T_i, v \in T_j$ there is no path between u and v in $G - S$. We remark that unless stated otherwise, a cut S might intersect the set T of terminals, respectively the set $\bigcup_{T \in \mathcal{T}} T$ in the general case. A set $S \subseteq V(G)$ is said to be *closest* to some set $T \subseteq V$ in G if S is the unique (T, S)-min cut in G, and hence, there exist $|S|$-many vertex-disjoint paths from T to S. A *T-closest cut* between T and S is a (T, S)-min cut that is closest to T. Kratsch and Wahlström [13] describe a simple efficient algorithm for computing a T-closest cut. In the minimization problem s-MULTIWAY CUT the value of a solution x for undirected graph G and set $T = \{t_1, \ldots, t_s\}$ of terminals, is equal to $|S|$ if x encodes a cut S for (G, T), which is furthermore disjoint from T, and $+\infty$ otherwise. In contrast to this, in the minimization problem DELETABLE TERMINAL MULTIWAY CUT we are given an arbitrarily large set T of terminals, but now x is allowed to encode a cut S for (G, T) which might intersect the terminal set T, in which case the value of x is also equal to $|S|$.

Parameterized complexity. A *parameterized problem* is a set $Q \subseteq \Sigma^* \times \mathbb{N}$. For a tuple $(x, \ell) \in \Sigma^* \times \mathbb{N}$ the number ℓ is called the *parameter*. For a problem Π (decision or optimization) and minimization problem ρ we define the *structurally parameterized* problem $\Pi[\rho]$, for which it holds that $(x, s, \ell) \in \Pi[\rho]$ if and only if s is a feasible solution of value at most ℓ for ρ on

x and: (i) for Π being a decision problem, $x \in \Pi$; (ii) for Π being a minimization problem, $x = (x', k)$ for some $k \in \mathbb{N}$ and $\mathrm{OPT}_\Pi(x') \leq k$; (iii) for Π being a maximization problem, $x = (x', k)$ and $\mathrm{OPT}_\Pi(x') \geq k$.

A *kernelization* for a parameterized problem Q is a polynomial-time algorithm $\mathcal{A}$ which is given as input an instance (x, ℓ) and outputs an *equivalent* instance (x', ℓ'), i.e., $(x, \ell) \in Q \Leftrightarrow (x', \ell') \in Q$, such that $|x'|$ and ℓ' are upper bounded by $f(\ell)$ for some computable function f. The function f is called the *size* of the kernelization $\mathcal{A}$, and if f is polynomially bounded, then $\mathcal{A}$ is called a *polynomial kernelization*. If there is a probability that the output instance (x', ℓ') is not equivalent to (x, ℓ), then $\mathcal{A}$ is called a *randomized kernelization*. In our cases, the error probability will be upper bounded by $\mathcal{O}(2^{-\Theta(|x|)})$. The output instance (x', ℓ') is called a *kernel* of (x, ℓ) and we say that the problem Q *admits a (randomized) (polynomial) kernel*, if there exists a (randomized) (polynomial) kernelization for Q.

Boundaried graphs and boundaried kernelization. A *boundaried graph* is a special kind of annotated graph $G_B = (G, B)$, where we call G the *underlying graph* and B the *boundary*. We define the earlier mentioned problems on boundaried graphs in the same way as on their underlying graphs, i.e., we ignore the boundary for these problems. For two boundaried graphs G_B, H_C we define the operation of *gluing* these graphs together, denoted by $G_B \oplus H_C$, which results in a new boundaried graph with boundary $B \cup C$ and consisting of the disjoint union of the vertex and edge sets of G and H, under the identification of the vertices in $B \cap C$. Throughout this paper we will tacitly assume that $V(G)$ and $V(H)$ intersect exactly at $B \cap C$, in which case the underlying graph of $G_B \oplus H_C$ can simply be seen as the union of each the vertex sets and the edge sets of G and H. Clearly, graph gluing is commutative and associative.

Based on regular graph gluing, we additionally define gluing of boundaried graphs with additional vertex annotations, which we will call *(vertex) annotated boundaried graphs*: For two such annotated boundaried graphs $(G_B, T_1, \ldots, T_r)$ and $(H_C, U_1, \ldots, U_r)$ for some $r \in \mathbb{N}$ and with $V(G) \cap V(H) \subseteq B \cap C$, the result of $(G_B, T_1, \ldots, T_r) \oplus (H_C, U_1, \ldots, U_r)$ is the annotated boundaried graph $(G_B \oplus H_C, W_1, \ldots, W_r)$ where $W_i = T_i \cup U_i$ for every $i \in [r]$.

We say that the instances $(G_B, \mathcal{T})$ and $(G'_B, \mathcal{T}')$ are *gluing equivalent* with respect to an optimization problem Π on annotated graphs and boundary B, denoted as $(G_B, \mathcal{T}) \equiv_{\Pi,B} (G'_B, \mathcal{T}')$, if there exists some constant $\Delta \in \mathbb{Z}$ such that for every boundaried graph H_B and tuple $\mathcal{U}$ of $V(H)$-subsets, it holds that $\mathrm{OPT}_\Pi((G_B, \mathcal{T}) \oplus (H_B, \mathcal{U})) = \mathrm{OPT}_\Pi((G'_B, \mathcal{T}') \oplus (H_B, \mathcal{U})) + \Delta$. If Π and B are clear from context, we will omit them and write $\equiv$ instead. The optimization problem Π has *finite integer index* if the number of equivalence classes of $\equiv_{\Pi,B}$ is upper bounded by some function $f(|B|)$. We refer to the full version of the paper for definitions of gluing equivalence and finite index for Π being a decision problem.

Let Π be an optimization or decision problem and ρ a minimization problem, both on annotated graphs. A *boundaried kernelization* for the parameterized problem $\Pi[\rho]$ is a polynomial-time algorithm $\mathcal{A}$ which is given as input an annotated boundaried graph $(G_B, \mathcal{T})$ and feasible solution s for ρ on $(G_B, \mathcal{T})$ together with the value ℓ of s; and $\mathcal{A}$ outputs an annotated boundaried graph $(G'_B, \mathcal{T}')$, such that $(G'_B, \mathcal{T}')$ is gluing equivalent to $(G_B, \mathcal{T})$ and the size of $(G'_B, \mathcal{T}')$ is bounded by $f(|B| + \ell)$ for some computable function f. Additionally, if Π is an optimization problem, then $\mathcal{A}$ also needs to output the offset Δ by which the optimum value for $(G_B, \mathcal{T}) \oplus (H_B, \mathcal{U})$ differs from that for $(G'_B, \mathcal{T}') \oplus (H_B, \mathcal{U})$ for any boundaried graph H_B and tuple $\mathcal{U}$ of $V(H)$-subsets. Analogously to (regular) kernelization, we define the size of $\mathcal{A}$ and the notions of a (randomized) (polynomial) boundaried kernelization and kernel. By minor adaptation of a result by Antipov and Kratsch [1, Lemma 4] and under the assumption

that Π and ρ are NP-optimization problems, the existence of a (randomized) (polynomial) boundaried kernelization for parameterized problem $\Pi[\rho]$ also implies the existence of a (randomized) (polynomial) kernelization for $\Pi[\rho]$. Additionally we define a boundaried kernelization for Π[local solution] to be a boundaried kernelization for Π parameterized by itself, i.e., the input is an annotated boundaried graph $(G_B, \mathcal{T})$ and a feasible solution s for Π on $(G_B, \mathcal{T})$, together with the value ℓ of s.

We will construct boundaried kernelization by the use of reduction rules. We say that a reduction rule is *gluing safe*, if it gets as input an instance $(G_B, \mathcal{T})$ and outputs a gluing equivalent instance $(G'_B, \mathcal{T}')$ and the corresponding shift Δ in solution value, if Π is an optimization problem and $\Delta \neq 0$.

▶ **Lemma 3** ([1, Lemma 1]). *Let Π be a pure graph problem, G and G' graphs, and B, C vertex subsets of both $V(G)$ and $V(G')$ such that $B \subseteq C$. Then $G_C \equiv_{\Pi,C} G'_C$ implies $G_B \equiv_{\Pi,B} G'_B$, with the same offset Δ for these two equivalences, if Π is an optimization problem.*

3 Matroid tools for kernelization

In this section we recall some of the results of Kratsch and Wahlström [13] regarding the use of matroids for polynomial kernelization, as well as the corresponding definitions.

A *matroid* is a pair $M = (E, \mathcal{I})$ of a *ground set* E and a collection of *independent sets* $\mathcal{I} \subseteq 2^E$, which are subsets of E, such that: (i) $\emptyset \in \mathcal{I}$; (ii) if $I_1 \subseteq I_2$ and $I_2 \in \mathcal{I}$, then also $I_1 \in \mathcal{I}$; and (iii) if $I_1, I_2 \in \mathcal{I}$ and $|I_2| > |I_1|$, then there exists some $x \in I_2 \setminus I_1$ such that $I_1 \cup \{x\} \in \mathcal{I}$. An independent set $B \in \mathcal{I}$ is called a *basis* of M if no superset of B is independent. One can also define matroid M by its set of bases. The *rank* $r(X)$ of a subset $X \subseteq E$ is the largest cardinality of an independent set $I \subseteq X$. The rank of M is $r(M) := r(E)$. Some matroid M is said to be *linear*, if there exists some $m \times n$ matrix over a field $\mathbb{F}$, such that $M = (E, \mathcal{I})$, where E is the set of columns of A, and $\mathcal{I}$ contains those subsets of E that are linearly independent over $\mathbb{F}$. In such a case we also say that A *represents* M, and A *is a representation of* M.

Let $D = (V, A)$ be a directed graph and let $S, T \subseteq V$. We say that T is *linked* to S in D if there exist $|T|$-many vertex-disjoint paths from S to T, also allowing paths of length zero. In particular, any set is linked to itself. Given a directed graph $D = (V, A)$, a set $S \subseteq V$ of source vertices, and a set $U \subseteq V$ of sink vertices, we define a special case of matroids, called *gammoid*, by the sets $T \subseteq U$ that are linked to S in D. We will only use a further special case thereof with $U = V$, called *strict gammoid*, and slightly abuse notation by simply calling these gammoids. Due to Perfect [15] and Marx [14], a representation of a (strict) gammoid can be computed in randomized polynomial time, with one-sided error that can be made exponentially small in the size of the graph.

Let $M = (V, \mathcal{I})$ be a matroid and let X be an independent set in M. A set $Y \subseteq V$ is said to *extend* X in M if it holds that $X \cap Y = \emptyset$ and $X \cup Y \in \mathcal{I}$. For a collection of V-subsets $\mathcal{Y} \subseteq 2^V$ we say that a subset $\mathcal{Y}' \subseteq \mathcal{Y}$ is r-representative for $\mathcal{Y}$ if for every set $X \subseteq V$ of size at most r, the existence of a set $Y \in \mathcal{Y}$ that extends X in M, implies the existence of a set $Y' \in \mathcal{Y}'$ that also extends X in M.

Next, we state a result by Marx [14] dubbed as the *representative sets lemma* by Kratsch and Wahlström, and follow that by results that build up on it.

▶ **Lemma 4** ([14, 13]). *Let M be a linear matroid of rank $r + s$, and let $\mathcal{Y}$ be a collection of independent sets of M, each of size s. There exists a set $\mathcal{Y}' \subseteq \mathcal{Y}$ of size at most $\binom{r+s}{s}$ that is r-representative for $\mathcal{Y}$. Furthermore, given a representation A of M, we can find such a set $\mathcal{Y}'$ in time $(m + ||A||)^{\mathcal{O}(s)}$, where $m = |\mathcal{Y}|$ and $||A||$ denotes the encoding size of A.*

▶ **Lemma 5** ([11, Theorem 5]). *Let M be a gammoid and let $\mathcal{A} = \{A_1, \ldots, A_m\}$ be a collection of independent sets, each of size p. We can find in randomized polynomial time a set $\mathcal{A}' \subseteq \mathcal{A}$ of size at most $\binom{p+q}{p}$ that is q-representative for $\mathcal{A}$.*

▶ **Lemma 6** ([13, Theorem 1.2]). *Let $G = (V, E)$ be a directed graph and let $S, T \subseteq V$. Let r denote the size of a minimum (S, T)-vertex cut (which may intersect S and T). There exists a set $Z \subseteq V$ with $|Z| = \mathcal{O}(|S| \cdot |T| \cdot r)$, such that for any $A \subseteq S$ and $B \subseteq T$, it holds that Z contains a minimum (A, B)-vertex cut as a subset. We can find such a set in randomized polynomial time with failure probability $\mathcal{O}(2^{-|V|})$.*

▶ **Lemma 7** ([13, Corollary 1.5]). *Let $G = (V, E)$ be an undirected graph and $X \subseteq V$. Let s be a constant. Then there is a set $Z \subseteq V$ of $\mathcal{O}(|X|^{s+1})$ vertices such that for every partition $\mathcal{X} = (X_0, X_X, X_1, \ldots, X_s)$ of X into $s + 2$ possibly empty parts, the set Z contains a minimum multiway cut of $(X_1, \ldots, X_s)$ in the graph $G - X_X$ as a subset. We can compute such a set in randomized polynomial time with failure probability $\mathcal{O}(2^{-|V|})$.*

Such a set Z, as described in the preceding two lemmas, is called a *cutset* for the pair of vertex sets (S, T) (Lemma 6), respectively for the vertex set X (Lemma 7).

The main operation for reducing our graphs will be that of *bypassing* a vertex v in a graph G (also called making vertex v *undeletable* in G). In this operation one removes the vertex v from G and adds shortcut edges between the neighbors of v. In other words, for all paths (u, v, w) in G, one adds the edge (u, w) or $\{u, w\}$ (depending on whether the graph is (un)directed or mixed) unless already present. Effectively, this operation forbids to take v into a cut, while preserving the separation achieved by any vertex cuts that avoid v. For more detail, see [13]. Under the term *bypassing a set* $W \subseteq V(G)$, we mean the repeated operation of bypassing vertices of W one after another, in any order. Observe that by repeated use, the following lemma also holds when bypassing a vertex set W, if $X \subseteq V \setminus W$.

▶ **Lemma 8** ([13, Proposition 2.3]). *Let $G = (V, E)$ be an undirected, directed, or mixed graph and let G' be the result of bypassing some vertex $v \in V$ in G. Then, for any set $X \subseteq V \setminus \{v\}$ and any vertices $s, t \in V \setminus (X \cup \{v\})$, there is a path from s to t in $G - X$ if and only if there is a path from s to t in $G' - X$.*

4 s-Multiway Cut[local solution]

As our first result, we construct a boundaried kernelization variant of the randomized polynomial kernel for the s-MULTIWAY CUT problem by Kratsch and Wahlström [13]. For this, we leverage the argument of Kratsch and Wahlström, that for any optimal solution X it holds that each neighbor of a terminal t is either contained in the solution or in the same connected component of $G - X$ as t. In the boundaried setting this is translated to the observation (for G_B, T_G being the given local graph and terminal set, H_B, T_H any possible "rest" of the global instance, and X an optimal solution) that additionally any boundary vertex $x \in B$ is contained either in the solution, shares the component of some terminal in $G_B \oplus H_B - X$, or is contained in a component without any terminals. Then, using Lemma 7, we can compute a set Z that contains the needed vertices for any multiway cut that corresponds to any combination of these restrictions, and bypass all vertices of $F \setminus Z$ with $F := V(G) \setminus B$. Furthermore, we substitute the size bound of $|N(T)|$ of Kratsch and Wahlström due to an LP argument by Guillemot [8], and instead we apply Reduction Rule 1.

▶ **Theorem 9** ($\star$). *In general, s-MULTIWAY CUT[local solution] does not admit a boundaried kernelization.*

▶ **Theorem 10.** *The parameterized problem* s-MULTIWAY CUT[local solution] *admits a randomized polynomial boundaried kernelization with* $\mathcal{O}((|B|+\ell)^{s+1})$ *vertices, with failure probability* $\mathcal{O}(2^{-|V|})$*, and under the restriction that externally glued instances do not contain* B*-vertices as terminals.*

As partly indicated above the input to the boundaried kernelization is a tuple (G_B, T_G, S, ℓ) such that $|T_G| \leq s$ and S is a multiway cut of size at most ℓ for T_G in G. In particular it follows that $B' := S \cup (B \setminus T_G)$ is also a multiway cut for T_G in G, i.e., every terminal $t \in T_G$ is contained in its own component C_t in $G - B'$.

▶ **Reduction Rule 1.** *For any* $t \in T_G$*, if* $|N(t)| > |B'|$*, then choose a minimum size cut* X *between* $N(t)$ *and* B' *(allowing terminal-deletion) which is closest to* $N(t)$*. Remove the edges incident with* t*, and make* X *the new neighborhood of* t*.*

▶ **Lemma 11** ($\star$)**.** *Reduction Rule 1 is gluing-safe.*

▶ **Reduction Rule 2.** *Assume that Reduction Rule 1 cannot be applied. Let* $Z \subseteq V(G)$ *be a vertex set such that for every partition* $\mathcal{X} = (X_0, X_X, X_1, \ldots, X_s)$ *of* $X := B \cup \bigcup_{t \in T_G} N_G(t)$ *into* $s+2$ *possibly empty parts, the set* Z *contains a minimum multiway cut of* $(X_1, \ldots, X_s)$ *in the graph* $G - X_X$*. Bypass every vertex from* $V(G) \setminus (Z \cup B \cup T_G \cup N_G(T_G))$*.*

▶ **Lemma 12.** *Reduction Rule 2 is gluing-safe.*

Proof. Let G_B denote the graph before applying the reduction rule, and G'_B the result thereof. Let H_B be the graph for gluing. Let T_G be the given set of terminals for G (resp. for G'), and $T_H \subseteq V(H) \setminus B$ the set of terminals given together with H_B. Let $T = T_G \cup T_H$ and assume without loss of generality that $|T| = s$. If $|T| < s$, we can add isolated terminals to H, while $|T| > s$ directly leads to $(G_B \oplus H_B, T)$ having no feasible solution. Likewise, we can assume that no two terminals are adjacent. Let $s_G = |T_G|$ and assume some arbitrary ordering of the terminals, such that the vertices of T_G can be denoted by $t_1, \ldots, t_{s_G}$ and those from T_H by $t_{s_G+1}, \ldots, t_s$.

"$\mathbf{OPT}(G_B \oplus H_B, T) \leq \mathbf{OPT}(G'_B \oplus H_B, T)$". Let Y' be a multiway cut for T in $G'_B \oplus H_B$. Since Y' does not contain any bypassed vertices, it follows from Lemma 8 that Y' is a multiway cut for T in $G_B \oplus H_B$.

"$\mathbf{OPT}(G_B \oplus H_B, T) \geq \mathbf{OPT}(G'_B \oplus H_B, T)$". Let Y be a multiway cut for T in $G_B \oplus H_B$ without terminal deletion. Thus, no component in $(G_B \oplus H_B) - Y$ contains more than one terminal and, obviously, every terminal $t \in T_G$ is in the same component as $N_G(t) \setminus Y$. Furthermore, any vertex in $B \setminus Y$ shares its component with at most one terminal. Based on this we partition the set X into $(X_0, X_X, X_1, \ldots, X_s)$ as follows: X_X contains those $B \setminus T$-vertices that are contained in Y and X_0 those that are not connected with any terminal in $(G_B \oplus H_B) - Y$; for each $t_i \in T$ put into X_i every $B \setminus Y$-vertex that is connected with t_i in $(G_B \oplus H_B) - Y$, and if further t_i is contained in T_G then also put all vertices from $N_G(t_i) \setminus Y$ into X_i.

Let $F = V(G) \setminus B$. Clearly, $Y \cap F$ is a multiway cut for $(X_1, \ldots, X_s)$ in $G - X_X$. By choice of Z, construction of G' and Lemma 8, there is also a multiway cut $Y^* \subseteq Z$ of size at most $|Y \cap F|$ for $(X_1, \ldots, X_s)$ in $G' - X_X$. We will now verify that $Y' := (Y \setminus F) \cup Y^*$ is a multiway cut for T in $G'_B \oplus H_B$ (without terminal deletion). For that, assume for contradiction that there is a path P between distinct terminals $t_{i_1}, t_{i_2} \in T$ in $(G'_B \oplus H_B) - Y'$. Note that no two terminals from T are connected in either $H - Y' = H - Y$ or $G' - Y'$. Thus, the path cannot go only through $H - B$ or $G' - B$ and contains vertices from $B \setminus (T \cup Y')$. Give these vertices an order along the path in the direction from t_{i_1} to t_{i_2},

i.e., $P = (t_{i_1}, \ldots, x_1, \ldots, x_2, \ldots, x_p, \ldots, t_{i_2})$, where $x_1, \ldots, x_p$ are the $B \setminus (T \cup Y')$-vertices contained in P and the vertices between x_i, x_j with distinct $i, j \in [p]$ are contained in $(V(G) \cup V(H)) \setminus (B \cup T \cup Y')$. If the path between x_i and x_j goes through $H - (B \cup Y')$, then they need to be in the same part of the earlier partition $(X_0, X_X, X_1, \ldots, X_s)$: Since it holds that $H - (B \cup Y') = H - (B \cup Y)$, the vertex x_i is connected to $t \in T$ in $(G_B \oplus H_B) - Y$ if and only if x_j is connected to t. However, if the path goes through $G' - (B \cup Y')$ and if, say, x_i is in the part X_h with $t_h \in T_H$, then it might also be the case that x_j is in the part X_0, since Y^* does not have to separate X_h-vertices from X_0-vertices. Furthermore, if terminal $t_h \in T$ is connected with x_i (either through $H - (B \cup Y')$ or $G' - (B \cup Y')$), then x_i needs to be contained in the part X_h. Combining these points we come to the fact that $x_1 \in X_{i_1}$ and $x_p \in X_{i_2}$, while at the same time it holds that $x_p \in X_{i_1} \cup X_0$ by the transversal from x_1 to x_p through $H - (B \cup Y')$ and $G' - (B \cup Y')$ between consecutive B-vertices x_i, x_{i+1} for every $i \in [p-1]$. A contradiction. ◀

As Reduction Rules 1 and 2 are clearly applicable (the former exhaustively) in polynomial time, it remains to find the cut cover Z for $B \cup \bigcup_{t \in T_G} N_G(t)$, for which we use Lemma 7. Altogether, this proves Theorem 10, with small failure chance due to the computation of Z.

5 Deletable Terminal Multiway Cut[local solution]

Quite closely related to the s-MULTIWAY CUT problem is DELETABLE TERMINAL MULTIWAY CUT, in which the number of input terminals is arbitrary, but one is also allowed to take them into the solution. This time, for their kernel, Kratsch and Wahlström [13] define an auxiliary graph G^*, which adds two sink-only copies v', v'' for each $v \in V(G)$ to graph G, i.e., $G^*[V(G)] = G$ and for each $v \in V(G)$ and each neighbor $u \in N_G(v)$, add the directed edges $(u, v'), (u, v'')$. They show that for any solution X for graph G and terminal set T, and any $x \in X$, there are $|X| + 2$ many vertex-disjoint paths from T to $X \cup \{x', x''\}$ in G^*. Using a slightly stronger formulation by Kratsch [11], we get a similar connectivity result between $B \cup T_G$ and $(X \cap F) \cup \{x', x''\}$ in G^*, this time with G_B being only the given local graph-part, T_G a subset of the terminals, and $F = V(G) \setminus B$. This allows us to leverage another argument of Kratsch and Wahlström, and show that for gammoid $\mathcal{M}$ on G^* with sources $B \cup T_G$, set $\mathcal{Z} = \{\{x, x', x''\} \mid x \in F\}$ and every $x \in (X \setminus T) \cap F$, it holds that $\{x, x', x''\}$ is contained in every $(2|B| + |T_G| - 1)$-representative set $\mathcal{Z}' \subseteq \mathcal{Z}$ on $\mathcal{M}$. Again, while Kratsch and Wahlström use an LP argument by Guillemot [8] in order to bound size of the terminal set by a linear factor on the sought solution size k, we go another route, as the LP argument does not work without knowing the whole instance. Namely, use argumentation due to Razgon [16] in order to remove isolated components and vertices that have very high connection to the terminal set. This will give a bound of $|T_G| \in \mathcal{O}((|B| + \ell)^2)$, where ℓ is the size of a given local DELETABLE TERMINAL MULTIWAY CUT solution S on (G, T_G).

▶ **Theorem 13.** *The parameterized problem* DELETABLE TERMINAL MULTIWAY CUT[local solution] *admits a randomized polynomial boundaried kernel with $\mathcal{O}((|B| + \ell)^6)$ vertices.*

▶ **Lemma 14 (⋆).** *There exists a set of polynomial-time exhaustively applicable and gluing safe reduction rules, after which it holds that $|T_G| \in \mathcal{O}((|B| + \ell)^2)$.*

▶ **Reduction Rule 3.** *Let $\mathcal{M}$ be the gammoid for G^* with sources $B \cup T_G$ and let $\mathcal{Z} := \{\{v, v', v''\} \mid v \in F\}$. Further, let $\mathcal{Z}'$ be a $(2|B| + |T_G| - 1)$-representative subset of $\mathcal{Z}$. Then bypass every vertex $v \in F \setminus T_G$ for which $\{v, v', v''\}$ is not contained in $\mathcal{Z}'$.*

▶ **Lemma 15.** *Reduction Rule 3 is gluing safe.*

Proof. Let G' be the resulting graph. Fix H_B, T_H. Let $T := T_G \cup T_H$.

"$\mathbf{OPT}(G_B \oplus H_B, T) \leq \mathbf{OPT}(G'_B \oplus H_B, T)$". Let X' be a solution for $(G'_B \oplus H_B, T)$. Obviously, X' does not contain any bypassed vertices and thus by Lemma 8 it is also a solution for $(G_B \oplus H_B, T)$.

"$\mathbf{OPT}(G_B \oplus H_B, T) \geq \mathbf{OPT}(G'_B \oplus H_B, T)$". Let X be a solution for $Q := G_B \oplus H_B$ with terminal set T, such that $|X|$ is minimized but among such $|X \cap T|$ is maximized. Let $X_F := X \cap F$, $X_0 := X \setminus T$, and $T_0 := T \setminus X$. Further, define the mixed graph Q^* on the base of Q, but with two additional sink-only copies v', v'' for every vertex $v \in V(Q)$. Kratsch [11] has shown that in $Q^* - (T \cap X)$ there exists a path packing with $|X_0| + 2$ paths from T_0 to $X_0 \cup \{x', x''\}$ for each $x \in X_0$. Note that this implies existence of a path packing in $G^* - (T \cap X)$ with $|X_0 \cap F| + 2$ paths from $(B \cup T_G) \setminus X$ to $(X_0 \cap F) \cup \{x', x''\}$ for each $x \in X_0 \cap F$: Take the path packing from T_0 to $X_0 \cup \{x', x''\}$ in $Q^* - (T \cap X)$, throw away any path going to $X_0 \setminus F$, and note that the remaining paths are either totally contained in $G^*[F \cup \{x', x''\}]$ or must intersect $B \setminus X$. For each path of the latter case we fix its last vertex u contained in $B \setminus X$ and use the subpath from u to $(X_0 \cap F) \cup \{x', x''\}$ for the path packing.

Now we show for any fixed $x \in X_0 \cap F$ that $\{x, x', x''\}$ is contained in $\mathcal{Z}'$. Fix the path packing of size $|X_0 \cap F| + 2$ from $(B \cup T_G) \setminus X$ to $(X_0 \cap F) \cup \{x', x''\}$ in $G^* - (T \cap X)$ and let U be the set of $B \setminus X$-vertices from which the paths go to $(X_0 \cap F) \cup \{x', x''\}$ without touching B any further. Set $Y := B \setminus U$ and observe that there exists a path packing of $|X_0 \cap F| + |Y| + 2$ paths from $B \cup T_G$ to $(X_0 \cap F) \cup Y \cup \{x', x''\}$ (including the length-0 path (y) for each $y \in Y$) in G^*, i.e., $\{x, x', x''\}$ extends $X_x := ((X_0 \cap F) - \{x\}) \cup Y$ in $\mathcal{M}$. Let us make sure that $|X_x|$ is upper bounded by $2|B| + |T_G| - 1$, thus leading to the fact that $\{x, x', x''\}$ is a correct candidate for $\mathcal{Z}'$: If it were true that $|X_x| > 2|B| + |T_G| - 1$, it would follow thereof that $|X_F| > |B| + |T_G|$ and thus that $\tilde{X} := (X \setminus V(G)) \cup B \cup T_G$ is a multiway cut for Q of smaller size, contradicting minimality of $|X|$. For this last point we remark that a potential path between terminals in $Q - \tilde{X}$ would need to be a path between terminals from $T_H \setminus B$ and contained entirely inside of $H - X$, contradicting the fact that X is a multiway cut for (Q, T).

Having seen that $\{x, x', x''\}$ is a candidate for $\mathcal{Z}'$ as an extension of X_x, we show that there is no other vertex $v \in F$ which extends X_x. Assume for contradiction that such a vertex v exists. First for any $v \in X_0 \setminus \{x\}$ it holds that $\{v, v', v''\} \cap X_x = \{v\} \neq \emptyset$ and thus that $\{v, v', v''\}$ does not extend X_x. Hence we consider the case that $v \in F \setminus X$. We remark that by choice of U we have that $U \cap X = \emptyset$, and that each of its vertices $u \in U$ is reachable from some terminal $t \in T$ in $Q - X$, with no two U-vertices sharing the terminal. Since by assumption there exist three paths from $B \cup T_G$ to $\{v, v', v''\}$ and at most one of them goes through x, two of these paths go directly through $G - X_x$. In particular this means that these two paths go from $U \cup T_G$ to $\{v, v', v''\}$, implying that v is connected with two different terminals in $Q - X$, a contradiction to X being a multiway cut for (Q, T).

As a result, for any fixed $x \in X_0 \cap F$ it must hold that $\{x, x', x''\}$ is contained in $\mathcal{Z}'$ and thus no X-vertices were bypassed while constructing G'. By Lemma 8 this means that X is also a multiway cut for $(G'_B \oplus H_B, T)$. ◀

Now, Theorem 13 is proven by first exhaustively applying the reduction rules behind Lemma 14 in order to bound $|T_G|$ by $\mathcal{O}((|B| + \ell)^2)$; and afterwards applying Lemma 5 and Reduction Rule 3 in order to compute the representative set $\mathcal{Z}' \subseteq \mathcal{Z}$ of size $\mathcal{O}((|B| + |T_G|)^3)$ and bypass all vertices $v \in F \setminus T_G$ for which $\{v, v', v''\}$ is not contained in $\mathcal{Z}'$. Together this yields our wanted vertex size of $\mathcal{O}((|B| + \ell)^6)$.

6 Odd Cycle Transversal[local]

In this section, we work with the problem ODD CYCLE TRANSVERSAL, which is not directly defined through cuts, other than the previous two problems. However, Reed et al. [17] have used a connection between odd cycle transversals and cuts, and this connection is what allows us to use representative sets on gammoids. Basically, given some odd cycle transversal X for graph G, and defining auxiliary graph G^* on G with dedicated vertices X^* corresponding to X-vertices, an optimal solution for G can be deduced by computing for each partition $S \cup T \cup R$ of X^* a minimum (S,T)-cut in $G^* - R$.

Kratsch and Wahlström [12] used this connection in order to get a polynomial compression in the form of a matroid with $\mathcal{O}(|X|)$ elements and randomized encoding size of $\mathcal{O}(|X|^3)$, which includes information on the min-cuts for each partition $S \cup T \cup R$ of X^*. Later the same authors showed a randomized polynomial kernelization for the problem Γ-FEEDBACK VERTEX SET$[k]$ [13], which includes ODD CYCLE TRANSVERSAL$[k]$ as a special case. A written out kernel specifically for this problem can be found in [7].

▶ **Theorem 16.** *The parameterized problem* ODD CYCLE TRANSVERSAL[local solution] *admits a randomized polynomial boundaried kernel with $\mathcal{O}((|B| + \ell)^6)$ vertices.*

Let us be given boundaried graph G_B and odd cycle transversal S for G. By Lemma 3 and since the class of bipartite graphs is hereditary, we can simply assume that S is a subset of B and redefine $B := B \cup S$ otherwise. Let $F = V(G) \setminus B$ and note that by previous assumption it holds that $G[F] = G - B$ is a bipartite graph. Fix some arbitrary bi-coloring (L, R) of $G[F]$ and define an auxiliary graph G^* with vertices $B_L \cup B_R \cup F$, where $B_{\texttt{side}} := \{x_{\texttt{side}} \mid x \in B\}$ for $\texttt{side} \in \{L, R\}$. The graph G^* contains edges such that $G^*[F] = G[F]$ and for all $x \in B$ vertex x_L is adjacent to $N_G(x) \cap R$, while x_R is adjacent to $N_G(x) \cap L$. In addition, for any pair $y, x \in B$ that are adjacent in G, add the edges $\{y_L, x_R\}$ and $\{y_R, x_L\}$ to G^*.

▶ **Lemma 17** (⋆). *Let G_B and G^* be as above and let H_B be one more boundaried graph. The size of a minimum odd cycle transversal for $G_B \oplus H_B$ is equal to the minimum over the sum of the sizes of an odd cycle transversal X for H and of a minimum cut between $B_\sim := (B_L \cap K_L^*) \cup (B_R \cap K_R^*)$ and $B_\Delta := (B_L \cap K_R^*) \cup (B_R \cap K_L^*)$ in $G^* - \{x_L, x_R \mid x \in B \cap X\}$, where (K_L, K_R) is an arbitrary bi-coloring of $H - X$ and $K^*_{side} := \{x_L, x_R \mid x \in B \cap K_{side}\}$ for $side \in \{L, R\}$.*

▶ **Reduction Rule 4.** *Let G_B, G^* and the corresponding vertex sets be as above. Let Z be a cut cover for G^* and the set $B_L \cup B_R$. Then remove all vertices in $F \setminus Z$ in G.*

For any two vertices u, v in $B \cup (F \cap Z)$, if there was an odd-length path between u and v with all inner vertices in $F \setminus Z$, then add an edge between u and v; and if there was such an even-length path between u and v, then add a path of length two between u and v. It might be the case that we add both an edge and a length-2 path between u and v. Also, for any $x \in B$, if there was an odd path consisting only of x and vertices from $F \setminus Z$, then add a triangle with x one of its vertices.

▶ **Lemma 18.** *Reduction Rule 4 is gluing-safe.*

Proof. Let G' be the resulting graph. Observe that $G' - B$ is bipartite as we only added independent edges for the triangles, substituted odd-length paths by edges, and even-length paths by length-2 paths. Construct the graph $\tilde{G}$ from G' in the same way as G^* was constructed from G. The following claim is analogous to Lemma 8 and used for our proof.

▷ Claim 19 (⋆). For any set $C \subseteq B_L \cup B_R \cup (F \cap Z)$ and any vertices $s, t \in B_L \cup B_R$, there is a path from s to t in $G^* - C$ if and only if there is a path from s to t in $\tilde{G} - C$.

We only show one direction of the proof here, as the other direction (to be found in the full version) works very similarly.

"$\mathbf{OPT}(G_B \oplus H_B) \leq \mathbf{OPT}(G'_B \oplus H_B)$". Let S' be an odd cycle transversal for $G'_B \oplus H_B$. By Lemma 17 we can divide this solution into an odd cycle transversal X' for H and a minimum size cut C' between $B_\sim$ and B_Δ in $\tilde{G} - \{x_L, x_R \mid x \in B \cap X'\}$. We can assume without loss of generality that none of the vertices created for the substituting paths and cycles are contained in C'. Hence C' only contains vertices from $B_L \cup B_R \cup (F \cap Z)$ and it follows from Claim 19 that C' is also a cut between $B_\sim$ and B_Δ in $G^* - \{x_L, x_R \mid x \in B \cap X'\}$. Hence by Lemma 17 there exists an odd cycle transversal of size at most $|S'|$ for $G_B \oplus H_B$. ◀

Hence, we get our randomized polynomial boundaried kernelization for Theorem 16 in the following way. Using Lemma 7, compute a cutset $Z \subseteq V(G^*)$ of size $\mathcal{O}((|B| + \ell)^3)$ for $B_L \cup B_R$. Afterwards apply Reduction Rule 4 in order to get a gluing equivalent graph G' with $\mathcal{O}((|B|+\ell)^3)$ vertices from $B \cup (F \cap Z)$ and at most $\mathcal{O}((|B|+\ell)^6)$ vertices for substituting odd cycles at vertices $x \in B$, and even-length paths between vertices from $B \cup (F \cap Z)$.

7 Vertex Cover[oct]

As our last problem, we consider VERTEX COVER[oct]. We only give a short overview here. For the complete set of reduction rules and used lemmas, we refer to the full version [2]. Similar to the previous section, we can assume to be given a boundaried graph G_B such that $G - B$ is bipartite. After some initial modifications on G_B, we perform some further reduction rules based on the increase of local solution size for $G - B$, when a certain subset $B' \subseteq B$ is explicitly forbidden, which is called the *conflict* caused by B' on F [9]. Namely, for some auxiliary graph G^*, we show a connection between conflicts in G and certain cuts in G^*, which will lead us to applying the representative sets framework in order to find a cut cover in G^*, using which we again choose vertices of G to bypass.

▶ **Theorem 20** ($\star$). *The parameterized problem* VERTEX COVER[oct] *admits a polynomial boundaried kernelization with* $\mathcal{O}((|B| + \ell)^3)$ *vertices.*

8 Conclusion

Boundaried kernelization is a recently introduced model for efficient local preprocessing due to Antipov and Kratsch [1]. That work gave polynomial boundaried kernelizations for several pure graphs problems, all based on local reduction rules, as well as several unconditional lower bounds. We showed that also global tools like the matroid-based approach of Kratsch and Wahlström [13] can be leveraged for boundaried kernelization. Moreover, this required to extend the underlying definitions to work for annotated graphs, e.g., graphs with a distinguished set of terminal vertices, including a natural generalization of gluing.

We think that this should also be possible for other problems covered by these tools, e.g., VERTEX COVER[$k - LP$], SUBSET FEEDBACK VERTEX SET[k], and GROUP FEEDBACK VERTEX SET[k], possibly with some restriction as was necessary for s-MULTIWAY CUT. Similarly, using a natural notion of boundaried formula, it would be interesting to get, if possible, a polynomial boundaried kernelization for ALMOST 2-SAT[k].

References

1 Leonid Antipov and Stefan Kratsch. Boundaried kernelization. *CoRR*, abs/2504.18476, 2025. To appear in proceedings of WG 2025. doi:10.48550/arXiv.2504.18476.

2 Leonid Antipov and Stefan Kratsch. Boundaried kernelization via representative sets. *CoRR*, abs/2510.00832, 2025. doi:10.48550/arXiv.2510.00832.

3 Hans L. Bodlaender, Fedor V. Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M. Thilikos. (Meta) kernelization. *J. ACM*, 63(5):44:1–44:69, 2016. doi:10.1145/2973749.

4 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.

5 Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, Berlin, Heidelberg, 2025. doi:10.1007/978-3-662-70107-2.

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

7 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: Theory of Parameterized Preprocessing*. Cambridge University Press, 2019.

8 Sylvain Guillemot. FPT algorithms for path-transversal and cycle-transversal problems. *Discrete Optimization*, 8(1):61–71, 2011. doi:10.1016/j.disopt.2010.05.003.

9 Bart M. P. Jansen and Hans L. Bodlaender. Vertex Cover Kernelization Revisited: Upper and Lower Bounds for a Refined Parameter. *Theory of Computing Systems*, 53(2):263–299, 2013. doi:10.1007/s00224-012-9393-4.

10 Bart M. P. Jansen and Stefan Kratsch. A structural approach to kernels for ILPs: Treewidth and total unimodularity. In Nikhil Bansal and Irene Finocchi, editors, *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, volume 9294 of *Lecture Notes in Computer Science*, pages 779–791. Springer, 2015. doi:10.1007/978-3-662-48350-3_65.

11 Stefan Kratsch. Recent developments in kernelization: A survey. *Bull. EATCS*, 113, 2014. URL: http://eatcs.org/beatcs/index.php/beatcs/article/view/285.

12 Stefan Kratsch and Magnus Wahlström. Compression via Matroids: A Randomized Polynomial Kernel for Odd Cycle Transversal. *ACM Trans. Algorithms*, 10(4):20:1–20:15, 2014. doi:10.1145/2635810.

13 Stefan Kratsch and Magnus Wahlström. Representative Sets and Irrelevant Vertices: New Tools for Kernelization. *J. ACM*, 67(3):16:1–16:50, 2020. doi:10.1145/3390887.

14 Dániel Marx. A parameterized view on matroid optimization problems. *Theoretical Computer Science*, 410(44):4471–4479, 2009. doi:10.1016/j.tcs.2009.07.027.

15 Hazel Perfect. Applications of Menger's graph theorem. *Journal of Mathematical Analysis and Applications*, 22(1):96–111, 1968. doi:10.1016/0022-247X(68)90163-7.

16 Igor Razgon. Large isolating cuts shrink the multiway cut. *CoRR*, abs/1104.5361, 2011. arXiv:1104.5361.

17 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.

18 Magnus Wahlström. Abusing the tutte matrix: An algebraic instance compression for the k-set-cycle problem. In Natacha Portier and Thomas Wilke, editors, *30th International Symposium on Theoretical Aspects of Computer Science, STACS 2013, February 27 - March 2, 2013, Kiel, Germany*, volume 20 of *LIPIcs*, pages 341–352. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013. doi:10.4230/LIPICS.STACS.2013.341.

Parameterized Complexity of Scheduling Unit-Time Jobs with Generalized Precedence Constraints

Christina Büsing
Research and Teaching Area for Combinatorial Optimization, RWTH Aachen University, Germany

Maurice Draeger
RWTH Aachen University, Germany

Corinna Mathwieser[1]
Research and Teaching Area for Combinatorial Optimization, RWTH Aachen University, Germany

Abstract

We study the parameterized complexity of scheduling unit-time jobs on parallel, identical machines under generalized precedence constraints for minimization of the makespan and the sum of completion times ($P|gen\text{-}prec,\ p_j = 1|\gamma,\ \gamma \in \{C_{\max}, \sum_j C_j\}$). In our setting, each job is equipped with a Boolean formula (precedence constraint) over the set of jobs. A schedule satisfies a job's precedence constraint if setting earlier jobs to true satisfies the formula. Our definition generalizes several common types of precedence constraints: classical *and*-constraints if every formula is a conjunction, *or*-constraints if every formula is a disjunction, and *and*/*or*-constraints if every formula is in conjunctive normal form. We prove fixed-parameter tractability when parameterizing by the number of predecessors. For parameterization by the number of successors, however, the complexity depends on the structure of the precedence constraints. If every constraint is a conjunction or a disjunction, we prove the problem to be fixed-parameter tractable. For constraints in disjunctive normal form, we prove $W[1]$-hardness. We show that the *and*/*or*-constrained problem is $\mathcal{NP}$-hard, even for a single successor. Moreover, we prove $\mathcal{NP}$-hardness on two machines if every constraint is a conjunction or a disjunction. This result not only proves para-$\mathcal{NP}$-hardness for parameterization by the number of machines but also complements the polynomial-time solvability on two machines if every constraint is a conjunction ([4]) or if every constraint is a disjunction ([11]).

2012 ACM Subject Classification Mathematics of computing → Combinatorial optimization

Keywords and phrases scheduling, precedence constraints, fixed-parameter tractability, complexity

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.7

1 Introduction

Scheduling jobs with precedence constraints on identical parallel machines is a well-researched problem with many real-world applications. A classical precedence constraint between two jobs, say from job i to job j, means that job j cannot begin until job i has finished. If job j has multiple such constraints, it can only start once all the predecessor jobs have been completed. However, many real-life settings find classical precedence constraints (*and*-type constraints) too restrictive. It may not always be necessary to complete all predecessors before starting a job. This need for flexibility has led to the proposal of alternative variants of precedence constraints in the literature. The first variant is *or*-precedence constraints where completing any predecessor is sufficient to start a job. The second variant includes *and*+*or*-constraints where some jobs require all predecessors to be completed earlier (*and*-jobs) while others only require to be preceded by only one predecessor (*or*-jobs). Another more general variant is known as *and*/*or*-constraints. In this model, each job has multiple groups of predecessor jobs, and to start the job, at least one job from each group must be completed first. A motivating

[1] corresponding author

© Christina Büsing, Maurice Draeger, and Corinna Mathwieser;
licensed under Creative Commons License CC-BY 4.0

example for *and*/*or*-constraints is provided by Goldwasser, Lotombe and Motwani ([7]), who examine the problem of disassembling a product. Certain parts may need to be taken out prior to accessing other components. Occasionally, the same part might be accessible from various directions and therefore, removing any of several different components may provide access to a particular part. Scheduling with a combination of *or*- and *and*-constraints has further been studied in the works of Gillies and Liu [6], Möhring et al. [17], Lee et al. [13] and Erlebach et al. [5]. For scheduling under solely *or*-constraints, we refer to the works of Berit [11] and the paper by Happach [10].

Given the diverse landscape of precedence constraint types, we propose a more general notion of precedence constraints, which encompasses all aforementioned types. Specifically, we encode precedence constraints regarding a specific job j as a logical formula ψ_j over (unnegated) variables representing the set J of jobs: $\{x_i \mid i \in J\}$. In any feasible schedule, ψ_j must be satisfied by the assignment resulting from setting $x_i = 1$ precisely if job i is executed prior to j. Our definition gives rise to *and*-constraints when each ψ_j is a conjunction; to *or*-constraints when each ψ_j is a disjunction; to *and*+*or*-constraints when each ψ_j is either; and to *and*/*or*-constraints when each ψ_j takes conjunctive normal form. Naturally, another interesting variant arises when each ψ_j is expressed in disjunctive normal form. We refer to this new variant as *or*/*and*-constraints. To gain an intuitive motivation of *or*/*and*-constraints, consider the following example from disaster management: There are several routes that can be used to evacuate people from a village, but obstacles (such as debris or fallen trees) are blocking these routes. To proceed with the evacuation, it might not be essential to clear all routes yet but it is necessary to remove all obstacles from one route.

The study of parameterized algorithms as an approach to addressing computationally hard problems is a vibrant field of research that has gained significant momentum in the scheduling community in recent years ([15], [19], [3], [1], [18], [16], [12]). In the context of classical precedence-constrained scheduling problems, the most common parameters include the number of machines, the width of the partial order that is induced by the precedence constraints, and the maximum processing time. Observe that the effect of precedence constraints on the complexity of scheduling problems can be twofold: On the one hand, the additional structure imposed by precedence constraints can simplify scheduling problems. An extreme example is when precedence constraints result in a total order among all jobs, rendering scheduling problems trivial. On the other hand, precedence constraints can also turn polynomial-time solvable scheduling problems into $\mathcal{NP}$-hard problems (Prot and Bellenguez-Morineau [19]): Scheduling unit-time jobs on parallel identical machines while minimizing the makespan or the (weighted) sum of completion times becomes $\mathcal{NP}$-hard in the presence of *and*-precedence constraints (Lenstra and Kan [14], Sethi [20]). Restricting the width of the partial order in classical precedence-constrained scheduling seeks to exploit the former effect: A small width generates high interrelatedness between the jobs and may thus provide scheduling problems with more structure. Yet, this approach does not necessarily result in fixed-parameter tractability, not even in combination with other classical parameters. Bodlaender and Fellows ([2]) show that even $P|prec,\ p_j = 1|C_{\max}$ is $W[2]$-hard when parameterized by the width and the number of machines and Van Bevern et al. show in [21] that $P2|prec,\ p_j \in \{1,2\}|C_{\max}$ is $W[2]$-hard when parameterized by the width. The diversity in the forms that precedence constraints can take – and the observation that enforcing dense dependency structures does not necessarily improve tractability – motivates the introduction of new parameters for generalized precedence constraints which limit interrelatedness.

Our Contribution

We study the problem $P|\textit{gen-prec},\ p_j = 1|\gamma$ of scheduling unit-time jobs on parallel, identical machines under generalized precedence constraints and consider the objectives of minimizing the makespan ($\gamma = C_{\max}$) or the sum of completion times ($\gamma = \sum_j C_j$). We investigate the parameterized complexity with respect to three parameters: the number k_p of predecessors, the number k_s of successors and the number m of machines. We achieve fixed-parameter tractability when parameterizing by the total number k_p of jobs that appear as predecessors in any precedence constraint, by providing a single-exponential algorithm. For the number k_s of successors however, the picture is more diverse and depends on the structure of the precedence constraints. For *and+or*-constraints, we prove the problem to be fixed-parameter tractable (FPT) by providing an algorithm with doubly-exponential dependence on k_s. This result does not extend to *and/or*- nor to *or/and*-constraints: We prove that $P|\textit{and/or-prec},\ p_j = 1|\gamma$ is para-$\mathcal{NP}$-hard with respect to k_s by showing that the problem is $\mathcal{NP}$-hard, even for a single successor. For *or/and*-constraints, we prove the problem to be in XP and to be $W[1]$-hard. Since Berit provides in [11] a polynomial-time algorithm for scheduling unit-time jobs on parallel, identical machines under *or*-constraints, it makes sense to ask if *and+or*-constrained scheduling remains FPT when only restricting the number $k_s^{and} \leq k_s$ of *and*-successors. However, we prove that scheduling under *and+or*-constraints is $\mathcal{NP}$-hard, even if there is only one *and*-job. Since our reduction only makes use of two machines, we also obtain that $P|\textit{and+or},\ p_j = 1|\gamma$ is para-$\mathcal{NP}$-hard with respect to the number m of machines.

2 Preliminaries

Problem Definition

We consider the problem of non-preemptively scheduling unit time jobs on identical, parallel machines under generalized precedence constraints. An instance specifies a set $J = \{1, \dots, n\}$ of jobs, a number m of identical machines and occasionally job weights $w_j \geq 0$, $j \in J$. A schedule $\mathcal{S}$ specifies which job, if any, each machine works on at each point in time. More precisely, a schedule $\mathcal{S}$ is represented as tuple $(C_j)_{j \in J}$, where $C_j \in \mathbb{N}$ indicates the *completion time* of each job $j \in J$. Since m jobs can be processed in parallel, we assume that $|\{j \in J \mid C_j = t\}| \leq m$ for any time $t \in \mathbb{N}$. We say that a job i *precedes* a job j in $\mathcal{S}$ if $C_i < C_j$. Moreover, we refer to the period between time $t-1$ and time t as *time slot* $t \in \mathbb{N}$.

Precedence constraints comprehend the requirement that certain jobs have to be finished before others can be started. In order to formalize precedence constraints, we introduce a Boolean variable $x_j \in \{0, 1\}$ for each job $j \in J$. Moreover, we denote by $\mathbb{1}$ the logical one that represents the truth value "true". We encode precedence constraints as family $(\psi_j)_{j \in J}$, where each ψ_j, $j \in J$, is a Boolean formula over the (unnegated) variables in $\{x_i \mid i \in J\} \cup \{\mathbb{1}\}$. If a variable x_i occurs in a precedence constraint ψ_j, $i, j \in J$, then i is called a *predecessor of* j and j is called a *successor of* i. A job $i \in J$, which is the predecessor of some other job, is called a *predecessor*. Likewise, a job $j \in J$ with non-trivial precedence constraint $\psi_j \neq \mathbb{1}$, is called a *successor*. We refer by k_p (k_s) to the number of predecessors (successors).

A schedule $\mathcal{S}$ is associated with variable assignments $x^j(\mathcal{S})$, $j \in J$, which set $x_i^j(\mathcal{S}) = 1$ precisely if i precedes j in $\mathcal{S}$. A schedule $\mathcal{S}$ is called *feasible* with respect to precedence constraints $(\psi_j)_{j \in J}$ if each ψ_j, $j \in J$, is satisfied by setting $x_i = x_i^j(\mathcal{S})$ for $i \in J$. Given a schedule $\mathcal{S}$, we call a job j *available* at time t if the variable assignment that sets $x_i = 1$ precisely if $C_i \leq t$ is a true assignment for ψ_j. We consider three classical objectives: Minimizing the makespan ($\gamma := C_{\max} = \max_{j \in J} C_j$), minimizing the sum of completion times

($\gamma := \sum_j C_j$) or minimizing the weighted sum of completion times, ($\gamma := \sum_j w_j C_j$). Following standard notation as introduced by Graham et al. in [9], we refer to the problem of finding an optimal schedule by $P|\textit{gen-prec},\ p_j = 1|\gamma$. Here, P means identical parallel machines, $p_j = 1$ indicates unit processing times for all jobs, and $\gamma \in \{C_{\max}, \sum_j C_j, \sum_j w_j C_j\}$ specifies the objective function. If we consider the problem for two instead of arbitrarily many machines, we write $P2|\textit{gen-prec},\ p_j = 1|\gamma$. Given an instance I of $P|\textit{gen-prec},\ p_j = 1|\gamma$, we denote the objective value of an optimal solution by $OPT(I)$ and of a specific schedule $\mathcal{S}$ by $\gamma(\mathcal{S})$. We distinguish five special cases of $P|\textit{gen-prec},\ p_j = 1|\gamma$, $\gamma \in \{C_{\max}, \sum_j C_j, \sum_j w_j C_j\}$:

- $P|\textit{prec},\ p_j = 1|\gamma$, where each ψ_j is a conjunction,
- $P|\textit{or-prec},\ p_j = 1|\gamma$, where each ψ_j is a disjunction,
- $P|\textit{and+or-prec},\ p_j = 1|\gamma$, where each ψ_j is either a conjunction or a disjunction,
- $P|\textit{and/or-prec},\ p_j = 1|\gamma$, where each ψ_j is in conjunctive normal form,
- $P|\textit{or/and-prec},\ p_j = 1|\gamma$, where each ψ_j is in disjunctive normal form.

Feasibility and objectives

A feasible schedule for $P|\textit{gen-prec},\ p_j = 1|\gamma$ can be calculated in polynomial time by repeatedly choosing an available job and assigning it to any free machine. For a precise description, we refer to Algorithm 1 by Möhring et al. [17], which was formulated for *and/or*-constraints but can be easily implemented for generalized precedence constraints. In case of classical *and*-constraints, infeasibility is equivalent to the existence of a cycle in the precedence graph G, where G is a digraph with node set J and arc set

$$A = \{(i,j) \mid i,j \in J,\ i \text{ is a predecessor of } j\}.$$

Throughout this paper, we assume that all instances are feasible.

Scheduling with unit-time jobs has the advantage that we can minimize the makespan and the sum of completion times simultaneously. While not every feasible schedule with minimum makespan necessarily optimizes the sum of completion times, the reverse holds true (which can be seen through standard swapping arguments). Subsequently, it suffices to only prove the correctness of algorithms for $\gamma = \sum_j (w_j) C_j$ and hardness for $\gamma = C_{\max}$.

List Scheduling

List scheduling is a simple and intuitive method for addressing scheduling problems and was introduced by Graham in [8]. The approach involves assigning a priority to each job, creating an ordered list based on these priorities. Jobs are then scheduled in the order they appear on this list. Whenever a machine is free, it is assigned the first unscheduled job from the list that is available to be processed at that time. List scheduling ensures machines are never left idle intentionally – idle time only occurs when no jobs are currently available for processing. This method can be efficiently implemented and runs in polynomial time.

3 Parameterization by the number of predecessors

In this section, we analyze the parameterized complexity regarding the overall number k_p of predecessors. Note that the parameters k_p and k_s are, in a sense, conceptually opposed to width. While a small width typically indicates that most jobs are highly interrelated, having few predecessors or successors often leads to many jobs being largely independent. A small width is therefore used to enforce structure in scheduling problems that are hard without precedence constraints. In contrast, parameterizing by k_p or k_s aims to capture the effect

of introducing a limited number of precedence constraints into problems that are efficiently solvable when no precedence constraints are present (as is the case for unit processing times). In the following, we prove that $P|\textit{gen-prec},\ p_j = 1|\gamma,\ \gamma \in \{C_{\max}, \sum_j C_j\}$ is FPT when parameterized by the number of predecessors (see Theorem 2). For minimizing the weighted sum of completion times, we prove the weaker result that $P|\textit{gen-prec},\ p_j = 1|\sum_j w_j C_j$ lies in $\mathcal{P}$ if k_p is bounded by a constant. In either case, we provide an algorithm that essentially explores all possibilities of positioning the predecessors in a schedule and later inserts all other jobs via List Scheduling. To test fewer possibilities, we argue that we may assume all predecessors to be processed before time step k_p if $\gamma \in \{C_{\max}, \sum_j C_j\}$.

▶ **Lemma 1.** *Let $I = (J, (\psi_j)_{j\in J}, m)$ be a feasible instance of $P|\textit{gen-prec},\ p_j = 1|\gamma,\ \gamma \in \{C_{\max}, \sum_j C_j\}$ with a set $J_p \subseteq J$ of k_p predecessors. Then there exists an optimal schedule $\mathcal{S}^*$ with completion times $(C_j^*)_{j\in J}$ such that $C_j^* \leq k_p$ for all $j \in J_p$.*

Proof. Let $\mathcal{S}^* = (C_j^*)_{j\in J}$ be a schedule with optimal sum of completion times. Assume $C_j^* > k_p$ for some $j \in J_p$. Let $t^* \leq k_p$ be the earliest time slot where no predecessor is processed. Choose a job $i \in J$ with $C_i^* = t^*$. By choice of t^*, i is not a predecessor job. Let h be an earliest predecessor with $C_h^* > t^*$. Construct a schedule $\mathcal{S}$ from $\mathcal{S}^*$ by swapping i and h. Clearly, the swap is feasible and does not alter the sum of completion times. Thus $\mathcal{S}$ is optimal too, while the earliest time slot t where no predecessor is processed has increased, $t > t^*$. Repeat until all completion times of predecessor jobs do not exceed k_p. ◀

The algorithm ALG-PRE for instances of $P|\textit{gen-prec},\ p_j = 1|\gamma,\ \gamma \in \{C_{\max}, \sum_j C_j, \sum_j w_j C_j\}$, with predecessors $J_p = \{i_1, \ldots, i_{k_p}\} \subseteq J$ proceeds as follows: Repeatedly pick a configuration $(C_{i_\ell})_{\ell=1}^{k_p}$ of predecessor completion times from $\mathcal{C} := \{1, \ldots, k_p\}^{k_p}$ if $\gamma \in \{C_{\max}, \sum_j C_j\}$ or from $\mathcal{C} := \{1, \ldots, n\}^{k_p}$ if $\gamma = \sum_j w_j C_j$. If $(C_{i_\ell})_{\ell=1}^{k_p}$ is a feasible schedule for J_p, insert all remaining jobs through List Scheduling in order of non-increasing weight and return the schedule with the smallest objective value.

▶ **Theorem 2.** *Let I be a feasible instance of $P|\textit{gen-prec},\ p_j = 1|\gamma,\ \gamma \in \{C_{\max}, \sum_j C_j, \sum_j w_j C_j\}$ with k_p predecessors. If $\gamma \in \{C_{\max}, \sum_j C_j\}$, then* ALG-PRE *returns in $\mathcal{O}(k_p^{k_p} \cdot m \cdot n^2)$ time the completion times of an optimal schedule $\mathcal{S}^*$. Otherwise,* ALG-PRE *returns in $\mathcal{O}(n^{k_p+2} \cdot m)$ time the completion times of an optimal schedule $\mathcal{S}^*$.*

Proof. Let $\mathcal{S}^* = (C_j^*)_{j\in J}$ be an optimal schedule, such that $(C_{i_1}^*, \ldots, C_{i_{k_p}}^*) \in \mathcal{C}$. (The existence of such $\mathcal{S}^*$ follows from Lemma 1). Consider the iteration of ALG-PRE where $C_j = C_j^*$ for all $j \in J_p$. Clearly, the algorithm computes a feasible schedule $\mathcal{S} = (C_j)_{j\in J}$. Denote by k_p^t the number of predecessor jobs processed during time slot $t \in \{1, \ldots, n\}$.

We prove the optimality of $\mathcal{S} = (C_j)_{j\in J}$ for $\gamma = \sum_j w_j C_j$, from which the statement for the other two objectives follows immediately. Assume the jobs in $J \setminus J_p$ have $1 \leq q \leq n - k_p$ different weights that we denote by $w^1 \geq w^2 \geq \ldots \geq w^q$. We consider the vector $z(t) = (z_1(t), \ldots, z_q(t)) \in \mathbb{N}^q$ where $z_i(t)$, $i \in \{1, \ldots, q\}$, denotes the number of jobs in $J \setminus J_p$ with weight w^i that $\mathcal{S}$ processes during time slot $t \in \mathbb{N}$. For $\mathcal{S}^*$, we define $z^*(t)$, $t \in \mathbb{N}$, analogously. If $z(t) = z^*(t)$ for all $t \in \mathbb{N}$, then $\sum_j w_j C_j = \sum_j w_j C_j^*$ and $\mathcal{S}$ is optimal. Otherwise, let $T := \min\{t \mid z(t) \neq z^*(t)\}$ be the smallest time slot for which $z(T), z^*(T)$ are not equal.

Assume first that $z(T)$ is lexicographically smaller than $z^*(T)$. Let $i \in \{1, \ldots, q\}$ be such that $z_s^*(T) = z_s(T)$ for $s < i$ and $z_i^*(T) > z_i(T)$. It holds that $\sum_{t=1}^T z_i(t) < \sum_{t=1}^T z_i^*(t)$. Subsequently, there exists a job $j \in J$ with weight $w_j = w^i$, such that $C_j^* \leq T$ but $C_j > T$. Observe that j is not a predecessor, otherwise $C_j = C_j^*$. Since $C_j^* \leq T$, job j is available at

time $T-1$ in $\mathcal{S}^*$ and thus also in $\mathcal{S}$. Since ALG-PRE sets $C_j > T$, the following holds: When j is considered, the algorithm has already occupied all machines during time slot T, namely with predecessor jobs or jobs with weight at least $w_j = w^i$. This yields a contradiction as

$$m = k_p^T + \sum_s z_s(T) = k_p^T + \sum_{s<i} z_s(T) + z_i(T) = k_p^T + \sum_{s<i} z_s^*(T) + z_i(T)$$
$$< k_p^T + \sum_{s<i} z_s^*(T) + z_i^*(T) \leq m.$$

Assume now that $z^*(T)$ is lexicographically smaller than $z(T)$ and let again $i \in \{1, \ldots, q\}$ be such that $z_s^*(T) = z_s(T)$ für $s < i$ while $z_i^*(T) < z_i(T)$. By analogous arguments, there exists a job $j \in J \setminus J_p$ with weight w^i such that $C_j \leq T$, $C_j^* > T$ and j is available at time $T-1$ in $\mathcal{S}^*$.

The optimal schedule $\mathcal{S}^*$ occupies all machines during time slot T with predecessor jobs or jobs of weight at least $w_j = w^i$: Otherwise, swapping j with a job of smaller weight or placing it on an idle machine during time slot T would yield a feasible solution with smaller weighted sum of completion times. As before,

$$m = k_p^T + \sum_{s<i} z_s^*(T) + z_i^*(T) < k_p^T + \sum_{s<i} z_s(T) + z_i(T) \leq m$$

yields a contradiction.

We conclude with a brief runtime analysis: We sort all non-predecessor jobs with respect to weight once ($\mathcal{O}(n \log n)$ time). Then the algorithm checks $|\mathcal{C}|$ configurations $(t_1, \ldots, t_{k_p}) \in \mathcal{C}$ of predecessor completion times. For each such configuration, the algorithm inserts $\mathcal{O}(n)$ jobs $j \in J \setminus J_p$. For each job $j \in J \setminus J_p$, it checks for $\mathcal{O}(n)$ time steps t whether j is available at time t. For any such time step t, the algorithm searches for an idle machine among m machines. Since, $|\mathcal{C}| \in \mathcal{O}(k^k)$ if $\gamma \in \{C_{\max}, \sum_j C_j\}$ and $|\mathcal{C}| \in \mathcal{O}(n^k)$ otherwise, the claimed asymptotic runtime follows. ◀

4 Parameterization by the number of successors

We investigate whether restricting the number of successors has the same effect as restricting the number of predecessors. However, it turns out that the parameterized complexity with respect to the number of successors depends on the structure of the precedence constraints. More precisely, we prove that $P|\textit{and+or-prec},\ p_j = 1|\gamma$, $\gamma \in \{C_{\max}, \sum_j C_j\}$ is FPT when parameterized with respect to the number k_s of successors. For the more general *or/and*-constraints, we prove that $P|\textit{or/and-prec},\ p_j = 1|\gamma$, $\gamma \in \{C_{\max}, \sum_j C_j\}$ is polynomial-time solvable if the number k_s of successors is bounded by a constant. We show that this result cannot be improved to fixed-parameter tractability by proving that $P|\textit{or/and-prec},\ p_j = 1|\gamma$, $\gamma \in \{C_{\max}, \sum_j C_j\}$ is $W[1]$-hard when parameterized by the number k_s of successors. For *and/or*-constraints, we prove para-$\mathcal{NP}$-hardness. Throughout this section, we assume $\gamma \in \{C_{\max}, \sum_j C_j\}$.

To prove that $P|\textit{and+or-prec},\ p_j = 1|\gamma$ is FPT with respect to k_s, we proceed as follows: 1. We provide an FPT algorithm AND-ALG-SUCC for instances of $P|\textit{prec},\ p_j = 1|\gamma$ with k_s successors. 2. We argue how to solve $P|\textit{and+or-prec},\ p_j = 1|\gamma$ by repeatedly calling AND-ALG-SUCC. Algorithm AND-ALG-SUCC is similar to ALG-PRE and explores all possibilities of positioning the successors in a schedule before inserting all other jobs. To avoid naïvely testing all $\mathcal{O}(n^{k_p})$ possible ways to schedule the successors, we use a similar result to Lemma 1: We may assume all successors to be processed within a time span of length k_s.

▶ **Lemma 3.** *Let $I = (J, (\psi_j)_{j\in J}, m)$ be a feasible instance of $P|\text{gen-prec}, p_j = 1|\gamma$, $\gamma \in \{C_{\max}, \sum_j C_j\}$ with k_s successors and let $J_s \subseteq J$ denote the set of successors. Then there exists an optimal schedule $\mathcal{S}^* = (C^*_j)_{j\in J}$, such that $C^*_j - C^*_i \leq k_s - 1$ for all $i, j \in J_s$.*

Proof. Consider an optimal schedule $\mathcal{S}^*$ with completion times $\bar{C}_\ell$, $\ell \in J$, and assume that $\bar{C}^s_{\max} - \bar{C}^s_{\min} > k_s - 1$ where $\bar{C}^s_{\min} = \min\{\bar{C}_\ell \mid \ell \in J_s\}$ is the completion time of an earliest successor and $\bar{C}^s_{\max} = \max\{\bar{C}_\ell \mid \ell \in J_s\}$ is the completion time of a latest successor. Let t^* be the earliest time slot with $\bar{C}^s_{\min} < t^* < \bar{C}^s_{\max}$ where no successor is processed. Pick a job $j \in J \setminus J_s$ with $C^*_j = t^*$. Let i be a successor with $C^*_i = t^* - 1$. We construct from $\mathcal{S}^*$ another feasible schedule $\mathcal{S}$ with completion times C_ℓ, $\ell \in J$, by swapping jobs i and j. Clearly, $\mathcal{S}$ is feasible and optimal and $C^s_{\max} := \max\{C_\ell \mid \ell \in J_s\} = \bar{C}^s_{\max}$. If $t^* - 1 = \bar{C}^s_{\min}$ and i is the only successor with $\bar{C}_i = t^* - 1$, then $C^s_{\min} := \min\{C_\ell \mid \ell \in J_s\} > \bar{C}^s_{\min}$. Else, if $t^* - 1 > \bar{C}^s_{\min}$ and i is again the only successor with $\bar{C}_i = t^* - 1$, then $C^s_{\min} = \bar{C}^s_{\min}$ but the earliest time slot $t \in (C^s_{\min}, C^s_{\max})$ without successors in $\mathcal{S}$ has decreased, $t < t^*$. Otherwise, we also get $C^s_{\min} = \bar{C}^s_{\min}$ while the overall number of time slots $t \in (C^s_{\min}, C^s_{\max})$ without successors in $\mathcal{S}$ has decreased. By repeating the argument, we obtain an optimal solution where all successors are processed within a time span of length k_s. ◀

Given a set $J_s \subseteq J$ of successors, the algorithm AND-ALG-SUCC repeatedly picks a configuration $(C_j)_{j\in J_s}$ from

$$\mathcal{C} := \{(t_j)_{j\in J_s} \in \{1, \dots, n\}^{k_s} \mid \forall i, j \in J_s : t_i - t_j \leq k_s - 1\},$$

such that $|\{j \in J_s \mid C_j = t\}| \leq m$ for all $t \in \{1, \dots, n\}$ and $t_i < t_j$ if i is a predecessor of j. Then the algorithm inserts the remaining jobs as follows: AND-ALG-SUCC iterates over all successors in order of non-decreasing completion times. For each successor $j \in J_s$, the algorithm assigns the earliest possible completion time to each predecessor. If some predecessor cannot be scheduled before j, the respective schedule of successors is discarded. In the end, all remaining jobs (which are neither successor nor predecessor) are inserted as early as possible.

▶ **Proposition 4.** *Let I be a feasible instance of $P|prec, p_j = 1|\gamma$, $\gamma \in \{C_{\max}, \sum_j C_j\}$ with k_s successors. Then* AND-ALG-SUCC *returns in $\mathcal{O}(k_s^{k_s} \cdot m \cdot n^3)$ time an optimal schedule $\mathcal{S}^*$.*

Proof. By Lemma 3, there exists an optimal schedule $\mathcal{S}^*$ with completion times C^*_j, $j \in J$ such that $(C^*_{i_1}, \dots, C^*_{i_{k_s}}) \in \mathcal{C}$. Consider the iteration of AND-ALG-SUCC where $C_j = C^*_j$ for all $j \in J_s$. Since $\mathcal{S}^*$ is a feasible schedule with identical successor completion times, there is enough space on the machines prior to each successor $j \in J_s$ for AND-ALG-SUCC to schedule all of j's predecessors. Hence, $(C^*_{i_1}, \dots, C^*_{i_{k_s}})$ is not discarded and AND-ALG-SUCC computes a feasible schedule $\mathcal{S}$. We proceed by proving that $\mathcal{S}$ is in fact optimal. We define $z(t)$ as the number of jobs in $\{j \in J \setminus J_s \mid C_j = t\}$, which $\mathcal{S}$ processes during time slot $t \in \mathbb{N}$. For $\mathcal{S}^*$, we define $z^*(t)$, $t \in \mathbb{N}$, analogously. If $z(t) = z^*(t)$ for all $t \in \mathbb{N}$, then $(C_{\max}, \sum_j C_j) = (C^*_{\max}, \sum_j C^*_j)$ and $\mathcal{S}$ is optimal. Otherwise, let $T := \min\{t \mid z(t) \neq z^*(t)\}$ be the smallest time slot for which $z(T), z^*(T)$ are not equal.

Assume first that $z(T) < z^*(T)$. It holds that $\sum_{t=1}^{T} z(t) < \sum_{t=1}^{T} z^*(t)$. Subsequently, there exists a job $j \in J$, such that $C^*_j \leq T$ but $C_j > T$. Observe that j is no successor, otherwise $C_j = C^*_j$. When AND-ALG-SUCC assigns a completion time to j, there is an idle machine during time slot T because $z(T) < z^*(T) \leq m$ and AND-ALG-SUCC assigns a completion time $C_j \leq T$ to j, contradiction. If $z(T) > z^*(T)$, there exists a job $j \in J \setminus J_s$, such that $C^*_j > T$ but $C_j \leq T$. The optimal schedule $\mathcal{S}^*$ has an idle machine during time slot T because $z^*(T) < z(T) \leq m$. Adapting the optimal solution such that $C^*_j = T$ does not

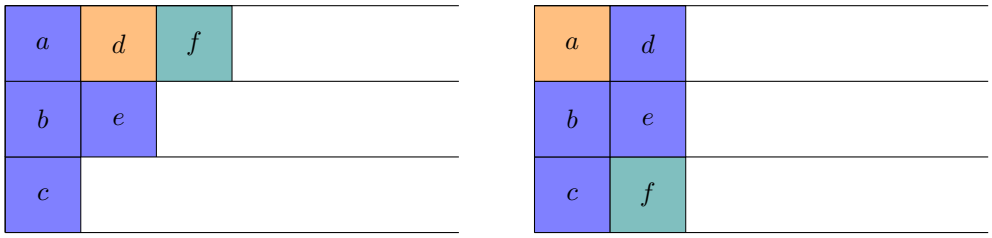


Figure 1 We process six jobs a, b, c, d, e, f with *and*+*or*-precedence constraints $\psi_e = x_a \wedge x_b \wedge x_c$, $\psi_f = x_a \vee x_d$ on three machines. The left schedule is optimal for the *and*-variation with $\psi'_f = x_d$. The schedule on the right is optimal for the *and*-variation with $\psi'_f = x_a$ and for the original instance.

negatively affect neither the optimality of $\mathcal{S}^*$ nor its feasibility since j has no predecessors. We repeat this argument until $z(t) = z^*(t)$ for all $t \in \mathbb{N}$, which proves that $\mathcal{S}$ is optimal. We omit a runtime analysis since it is analogous to the proof of Theorem 2. ◀

Using AND-ALG-SUCC, it is easy to see that $P|or/and\text{-}prec,\ p_j = 1|\gamma$, $\gamma \in \{C_{\max}, \sum_j C_j\}$, lies in $\mathcal{P}$ if the number of successors $k_s \leq const$ is bounded by some constant $const \geq 0$: Recall that *or*/*and*-constraints require each precedence constraint ψ_j, $j \in J$, to be a disjunction of conjunctive clauses. By electing one clause per job, we can solve an instance I of $P|or/and\text{-}prec,\ p_j = 1|\gamma$ by solving $\mathcal{O}(|I|^{k_s})$ instances of $P|prec,\ p_j = 1|\gamma$ with at most k_s successors. We state this in the following corollary:

▶ **Corollary 5.** *$P|or/and\text{-}prec,\ p_j = 1|\gamma$ can be solved in polynomial-time if the number of successors $k_s \leq const$ is bounded by a constant $const \geq 0$.*

For *and*+*or*-constraints, we can strengthen this argument to prove that $P|and{+}or\text{-}prec,\ p_j = 1|\gamma$, $\gamma \in \{C_{\max}, \sum_j C_j\}$ is FPT. Consider an instance $I = (J, (\psi_j)_{j \in J}, m)$ of $P|and{+}or\text{-}prec,\ p_j = 1|\gamma$ with k_s successors. Let $J_s = J_s^{and} \dot\cup J_s^{or}$ be a partition of the set of successors, where a precedence constraint ψ_j of a job $j \in J_s$ is a conjunction if $j \in J_s^{and}$ and a disjunction if $j \in J_s^{or}$. We call jobs in J_s^{and} *and*-jobs and jobs in J_s^{or} *or*-jobs. Consider an *or*-job $j \in J_s^{or}$ and let $i \in J$ be a predecessor of j. Consider the alternate instance $I' = (J, (\psi'_j)_{j \in J}, m)$ that we obtain by fixing i as only predecessor of j, that is, $\psi'_\ell = \psi_\ell$ if $\ell \neq j$ and $\psi'_j = x_i$. Clearly, any optimal schedule $\mathcal{S}'$ for I' is feasible for I too. However, whether $\mathcal{S}'$ is optimal for I too, strongly depends on whether i is also a predecessor of other jobs in $J_s \setminus \{j\}$. See Figure 1 for an example. Consider an arbitrary feasible schedule $\mathcal{S}$ for I. For each *or*-job $j \in J_s^{or}$, we can determine a predecessor job i_j of j such that i_j precedes j in $\mathcal{S}$. Observe that we can possibly have $i_j = i_k$ for two different *or*-jobs $j, k \in J_s^{or}$, $j \neq k$, and that each i_j may be a predecessor of multiple *and*-successors. In other words, the choice of $(i_j)_{j \in J_s^{or}}$ gives rise to a function $\sigma : J_s^{or} \to 2^{J_s}$ where $k \in \sigma(j)$ precisely if k is an *or*-job and $i_j = i_k$ or if k is an *and*-job with predecessor i_j. Our algorithm essentially tests all relevant variations of σ and picks an *and*-predecessor $i_j \in J$ for each *or*-job $j \in J_s$ according to σ, before it applies AND-ALG-SUCC to the resulting instance with *and*-constraints.

▶ **Definition 6.** *Let $I = (J, (\psi_j)_{j \in J}, m)$ be an instance of $P|and{+}or\text{-}prec,\ p_j = 1|\gamma$, $\gamma \in \{C_{\max}, \sum_j C_j\}$ where $J_s = J_s^{or} \dot\cup J_s^{and}$ is the set of successors. For each or-job $j \in J_s^{or}$, let $i_j \in J$ be a predecessor of j.*

1. *Let $\sigma : J_s^{or} \to 2^{J_s}$ be a function such that $k \in \sigma(j)$, $j \in J_s^{or}$, precisely if $k \in J_s^{or}$ and $i_j = i_k$ or if $k \in J_s^{and}$ with predecessor i_j. We call σ the* common predecessor function *of I with respect to $(i_j)_{j \in J_s^{or}}$.*

2. *If $\sigma : J_s^{or} \to 2^{J_s}$ is a common predecessor function of I with respect to $(i_j)_{j\in J_s^{or}}$, we call* $(i_j)_{j\in J_s^{or}}$ generating representatives *of σ.*
3. *Let $I' = (J, (\psi'_j)_{j\in J})$ be the $P|prec,\ p_j = 1|\gamma$ instance with $\psi'_j = \psi_j$ if $j \notin J_s^{or}$ and $\psi_j = x_{i_j}$ if $j \in J_s^{or}$. We call I' the* and-variation *of I with respect to $(i_j)_{j\in J_s^{or}}$.*

Observe the following properties of a common predecessor function $\sigma : J_s^{or} \to 2^{J_s}$: First, $j \in \sigma(j)$ for all $j \in J_s^{or}$. If $j, k \in J_s^{or}$ and $k \in \sigma(j)$, then $\sigma(j) = \sigma(k)$. If σ is the common predecessor function of I with respect to $(i_j)_{j\in J_s^{or}}$, then i_j is a predecessor of every $k \in \sigma(j)$, $j \in J_s^{or}$. Lastly, note that σ may have different generating representatives $(i_j)_{j\in J_s^{or}}$, $(i'_j)_{j\in J_s^{or}}$, $(i_j)_{j\in J_s^{or}} \neq (i'_j)_{j\in J_s^{or}}$. The following lemma proves that for a given common predecessor function, the choice of generating representatives is irrelevant (under certain conditions).

▶ **Lemma 7.** *Let $I = (J, (\psi_j)_{j\in J}, m)$ be a $P|and{+}or\text{-}prec,\ p_j = 1|\gamma$ instance, $\gamma \in \{C_{\max}, \sum_j C_j\}$, with successor set $J_s = J_s^{or} \dot\cup J_s^{and}$. Let $\sigma, \sigma' : J_s^{or} \to 2^{J_s}$ be common predecessor functions of I with respect to $(i_j)_{j\in J_s^{or}}, (i'_j)_{j\in J_s^{or}}$ respectively. Let L, L' be the and-variations of I with respect to $(i_j)_{j\in J_s^{or}}, (i'_j)_{j\in J_s^{or}}$ respectively. If*

$$I_s := \{j \in J_s^{or} \mid i_j \in J_s\} = \{j \in J_s^{or} \mid i'_j \in J_s\} \text{ and } i_j = i'_j \text{ for all } j \in I_s,$$

then L is feasible precisely if L' is feasible. In this case, $OPT(L') = OPT(L)$ if moreover $\sigma = \sigma'$ and $i_j = i'_k$ whenever $i'_j = i_k$, $j, k \in J_s^{or}$.

Proof. Assume that L is infeasible and let $C = (j_1, \dots, j_r)$ be a cycle in the precedence graph, $j_0 := j_r$. Consider a job j_ℓ in the cycle, $\ell \in \{1, \dots, r\}$. If $j_\ell \in J_s^{and}$ is an *and*-job in I, then $j_{\ell-1}$ is also a predecessor of j_ℓ in L'. If $j_\ell \in J_s^{or}$, then $j_{\ell-1} = i_j$. Since $j_{\ell-1}$ is a successor in L, it is also a successor in I and thus $j_{\ell-1} = i_j = i'_j$ is also a predecessor of j_ℓ in L'. Therefore, the precedence graph of L' contains the same cycle C and L' is infeasible.

Assume now that L is feasible, $\sigma = \sigma'$ and $i_j = i'_k$ whenever $i'_j = i_k$, $j, k \in J_s^{or}$. Let $(C_j)_{j\in J}$ be the completion times of an optimal schedule $\mathcal{S}$. We define a schedule $\mathcal{S}'$ with completion times $(C'_j)_{j\in J}$ for L'. We denote the sets in $\{\sigma(j) \cap J_s^{or} \mid j \in J_s^{or}\}$ by $S_1, \dots, S_r$. Observe that $(S_\ell)_{\ell=1}^r$ is a partition of J_s^{or} and that $S_\ell = \sigma(j) \cap J_s^{or}$ precisely if $S_\ell = \sigma'(j) \cap J_s^{or}$, $j \in J_s^{or}, \ell = 1, \dots, r$. For each $\ell = 1, \dots, r$, let $i(\ell),\ i'(\ell) \in J_s^{or}$ be the common predecessor with $i(\ell) = i_k,\ i'(\ell) = i'_k$ for all $k \in S_\ell$. We obtain $\mathcal{S}'$ from $\mathcal{S}$ by swapping the positions of $i(\ell)$ and $i'(\ell)$ for all $\ell = 1, \dots, r$. Observe that this is in fact the same as setting $C'_{i'_k} = C_{i_k}$ and $C'_{i_k} = C_{i'_k}$ for $k \in J_s^{or}$. All other jobs $j \in J$ remain as in $\mathcal{S}$, i.e. $C'_j = C_j$. Note that the schedule $\mathcal{S}'$ is well-defined, since we require $i'(\ell) = i(p)$ if $i(\ell) = i'(p)$ for $\ell, p = 1, \dots, r$. Clearly, $\gamma(\mathcal{S}) = \gamma(\mathcal{S}')$. It remains to show that $\mathcal{S}'$ is feasible. To this end, let $k \in J_s$ be a successor job in I. Observe that $C_k = C'_k$: If $k \neq i_\ell, i'_\ell$ for all $\ell \in J_s^{or}$, then $C'_k = C_k$ by definition of C_k. Otherwise, if $k = i_\ell$ or $k = i'_\ell$ for some $\ell \in J_s^{or}$, then $\ell \in I_s$ and thus $k = i_\ell = i'_\ell$ and $C'_k = C_k$. Assume now that $k \in J_s^{and}$ is an *and*-job in I and let $j \in J$ be a predecessor of k in L'. Clearly, j is also a predecessor of k in L and thus $C_j < C_k$. If $C'_j = C_j$, then j precedes k in $\mathcal{S}'$ because $C'_j = C_j < C_k = C'_k$. Otherwise, $j = i_\ell$ or $j = i'_\ell$ for some $\ell \in J_s^{or}$ with $i_\ell \neq i'_\ell$. Assume first that $j = i_\ell$. Since k is an *and*-job in I with predecessor $i_\ell = j$, it follows that $k \in \sigma(\ell) = \sigma'(\ell)$. By definition of σ', job i'_ℓ is a predecessor of k in I and L and thus $C'_j = C_{i'_\ell} < C_k = C'_k$. We can show analogously that $C'_j < C'_k$ if $j = i'_\ell$. Finally, let $k \in J_s^{or}$ be a former *or*-job. By definition of L', the job i'_k is the only predecessor of k in L' and i_k is the only predecessor of k in L. Subsequently, $C'_{i'_k} = C_{i_k} < C_k = C'_k$, which completes the proof. ◀

We now describe the algorithm AND+OR-ALG-SUCC for $P|and{+}or\text{-}prec,\ p_j = 1|\gamma$, $\gamma \in \{C_{\max}, \sum_j C_j\}$, in more detail. First, AND+OR-ALG-SUCC picks completion times C_j, $j \in J_s$, for the successors in the same way as AND-ALG-SUCC. For every *or*-job $j \in J_s^{or}$ with a predecessor $i \in J_s$ with $C_i < C_j$, we set $i_j = i$. Afterwards, AND+OR-ALG-SUCC iterates over all *or*-jobs $j_1, \ldots, j_q \in J_s^{or}$ in order of non-decreasing completion time $C_{j_1} \leq \ldots \leq C_{j_q}$. If no i_{j_i} has been chosen yet, we pick a subset $S_j \subseteq \{j_i, \ldots, j_q\} \cup J_s^{and}$ of successor jobs with $j \in S_j$. Then we pick a job $i \in J$ such that the successors of i in $\{j_i, \ldots, j_q\} \cup J_s^{and}$ are precisely the vertices in S_j. We set $i_k = i$ for all $k \in S_j \cap J_s^{or}$. (If such a job i does not exist, S_j is an infeasible choice.) Finally, the algorithm AND+OR-ALG-SUCC proceeds along the same lines as AND-ALG-SUCC for the *and*-variation of I w.r.t. $(i_j)_{j \in J_s^{or}}$.

▶ **Theorem 8.** *Let I be a feasible instance of $P|and{+}or\text{-}prec,\ p_j = 1|\gamma$, $\gamma \in \{C_{\max}, \sum_j C_j\}$ with k_s successors. Then* AND+OR-ALG-SUCC *returns in $\mathcal{O}(k_s^{k_s} \cdot 2^{k_s^2} \cdot m \cdot n^3)$ time the completion times of an optimal schedule $\mathcal{S}^*$.*

Proof. Let $J_s = J_s^{or} \dot\cup J_s^{and}$ be the set of successors with $|J_s| = k_s$ and $q := |J_s^{or}|$. By Lemma 3, there exists an optimal schedule $\mathcal{S}^* = (C_j^*)_{j \in J}$ such that $(C_{i_1}^*, \ldots, C_{i_{k_p}}^*) \in \mathcal{C}$. Consider the iteration of AND+OR-ALG-SUCC where $C_j = C_j^*$ for all $j \in J_s$ and let $\mathcal{S}$ denote the schedule defined by $(C_j)_j$. Let $j_1, \ldots, j_q$ be the ordering, in which AND+OR-ALG-SUCC iterates over the *or*-jobs. We denote by I_s the set of *or*-jobs $j \in J_s^{or}$, to which AND+OR-ALG-SUCC assigns a predecessor $i_j \in J_s$.

The idea of the proof is the following: First, we pick $(i_j^*)_{j \in J_s^{or}}$ such that i_j^* is a predecessor of j with $C_{i_j^*}^* < C_j^*$, $j \in J_s^{or}$. We argue, that for a certain choice of $(S_j)_j$, the algorithm computes representatives $(i_j)_{j \in J_s^{or}}$ such that $(i_j)_j, (i_j^*)_j$ as well as the associated common predecessor functions σ, σ^* comply with the conditions in Lemma 7.

More precisely, we pick $(i_j^*)_{j \in J_s^{or}}$ as follows: Consider the *or*-jobs j in order $j = j_1, \ldots, j_q$. We set $i_j^* = i_j$ for $j \in I_s$. If $j = j_i$ and no i_j^* has been chosen yet, we pick an arbitrary predecessor i_j^* with $C_{i_j^*}^* < C_j^*$ but set $i_k^* = i_j^*$ for all $k = j_{i+1}, \ldots, j_q$, of which i_j^* is also a predecessor. We denote by $\sigma^* : J_s^{or} \to 2^{J_s}$ the common predecessor function of I w.r.t. $(i_j^*)_{j \in J_s^{or}}$ and consider the iteration where AND+OR-ALG-SUCC sets $S_{j_i} = \sigma^*(j_i) \setminus \{j_1, \ldots, j_{i-1}\}$ for $i = 1, \ldots, q$. Observe that whenever AND+OR-ALG-SUCC is required to pick i_j according to S_j, then $S_j = \sigma^*(j)$, otherwise i_j would have been chosen in an earlier iteration. Due to the choice of $(S_j)_j$ and the properties of $(i_j^*)_j$, AND+OR-ALG-SUCC can find an i_j for $j = j_r$ such that 1. i_j is a predecessor of all jobs in S_j and 2. i_j is not a predecessor of any job in $(J_s^{and} \cup \{j_{r+1}, \ldots, j_q\}) \setminus S_j$. We denote by $\sigma : J_s^{or} \to 2^{J_s}$ the common predecessor function of I w.r.t. $(i_j)_{j \in J_s^{or}}$. Then it is easy to verify that $(i_j)_j, (i_j^*)_j, \sigma, \sigma^*$ comply with the conditions in Lemma 7, which completes the proof. ◀

We have shown that $P|or/and\text{-}prec,\ p_j = 1|\gamma$ is polynomial-time solvable if the number k_s of successors is bounded by a constant while $P|and{+}or\text{-}prec,\ p_j = 1|\gamma$ is even FPT with respect to the number of successors. Recall that Berit [11] proved the *or*-constrained problem $P|or,\ p_j = 1|\gamma$ to be polynomial-time solvable, even if no restrictions are made on the number of successors. Naturally, three questions arise:

1. Is $P|or/and\text{-}prec,\ p_j = 1|\gamma$ FPT when parameterized by k_s? We give a negative answer to this question by proving that $P|or/and\text{-}prec,\ p_j = 1|\gamma$ is $W[1]$-hard (see Theorem 9).
2. Is $P|and/or\text{-}prec,\ p_j = 1|\gamma$ polynomial-time solvable if the number k_s of successors is bounded by a constant? We will give a negative answer to this question by proving that $P|and/or\text{-}prec,\ p_j = 1|\gamma$ is $\mathcal{NP}$-hard, even if $k_s = 1$ (see Proposition 11).

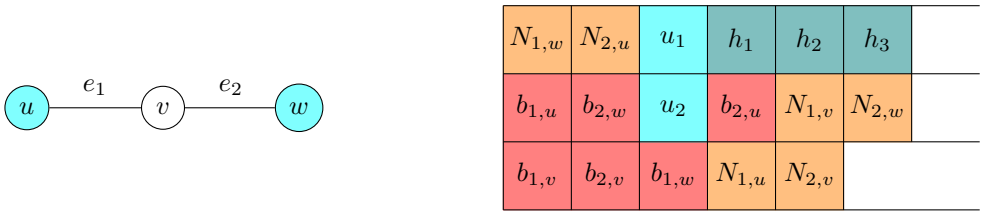


Figure 2 Reduction from Independent Set with $k = 2$ to $P|or/and\text{-}prec, p_j = 1|\gamma$ with $k_s = 2k+1$. Blue jobs and preceding orange jobs represent the size and selection of an independent set. In schedules with $C_{max} = 2k + 1$, green (red) jobs are used to block space after (before) blue jobs.

3. Is $P|and{+}or\text{-}prec,\ p_j = 1|\gamma$ FPT with respect to the number of *and*-successors? Surprisingly, $P2|and{+}or\text{-}prec,\ p_j = 1|\gamma$ turns out to be para-$\mathcal{NP}$-hard when parameterized by the number of *and*-successors. In fact, we prove that the problem is $\mathcal{NP}$-hard, even if there is only a single *and*-successor (see Theorem 13).

▶ **Theorem 9.** *$P|or/and\text{-}prec,\ p_j = 1|\gamma$, $\gamma \in \{C_{\max}, \sum_j C_j\}$ is $W[1]$-hard when parameterized by the number k_s of successors.*

Proof. We prove the claim by an FPT-reduction from Independent Set. Let $I' = (G, k)$ be an instance of Independent Set where $G = (V, E)$ is an undirected graph with $n' := |V|$ nodes and $k \in \mathbb{N}_0$ is the required size of an independent set, w.l.o.g. $k < n'$. By

$$\overline{\mathcal{N}}(v) := \{w \in V \mid \{v, w\} \in E\} \cup \{v\}$$

we denote the closed neighborhood of a vertex $v \in V$. We construct from I' an instance I of $P|or/and\text{-}prec,\ p_j = 1|C_{\max}$ with n' machines and require a makespan of at most $2(k + 1)$. An interpretation of the construction is given below. We introduce $2(k + k \cdot n') + 1$ jobs by setting

$$J := \{u_i \mid i = 1, \dots, k\} \cup \{N_{iv}, b_{iv} \mid i = 1, \dots, k, v \in V\} \cup \{h_i \mid i = 1, \dots, k + 1\}.$$

The precedence constraints are defined as follows: Jobs N_{iv}, b_{iv} in the second subset do not have predecessors. The auxiliary jobs $h_1, \dots, h_{k+1}$ have to be scheduled in a chain that follows the completion of all u_i, $i = 1, \dots, k$. That is, $\psi_{h_{i+1}} = x_{h_i}$ for all $i = 1, \dots, k$ and $\psi_{h_1} = \bigwedge_{i=1}^{k} x_{u_i}$. The precedence constraint of a job u_i is the disjunction $\psi_{u_i} = \bigvee_{v \in V} K_{iv}$ of n' clauses K_{iv}, where

$$K_{iv} := \bigwedge_{u \notin \overline{\mathcal{N}}(v)} x_{N_{iu}} \wedge \bigwedge_{u \in \overline{\mathcal{N}}(v)} x_{b_{iu}} \wedge \bigwedge_{j<i} x_{N_{jv}},\ i = 1, \dots, k, v \in V.$$

We interpret $v \in V$ as the i-th node in the independent set if K_{iv} is satisfied by the jobs that precede u_i, $i = 1, \dots, k$. In a schedule with makespan $2(k + 1)$, the chain of the auxiliary jobs $h_1, \dots, h_{k+1}$ guarantees that all jobs $u_1, \dots, u_k$ have to be completed by time $k + 1$. This leaves a total of at most $k \cdot n'$ time slots to schedule the predecessors that are needed to make $u_1, \dots, u_k$ available. However, each clause K_{iv} is composed of $n' + i - 1$ literals in a way that each of the last $i - 1$ predecessors N_{jv}, $j = 1, \dots, i - 1$, has to be a common predecessor with u_j, due to the space restrictions. Due to this, a node v_i for which K_{iv_i} is satisfied is disjoint and non-adjacent to the nodes v_j for which the previous $i - 1$ clauses K_{jv_j} are satisfied, $j = 1, \dots, i - 1$. Figure 2 depicts an example of the construction. Since an optimal schedule for $\gamma = \sum_j C_j$ simultaneously optimizes the makespan, the hardness for the sum of completion times follows too. ◀

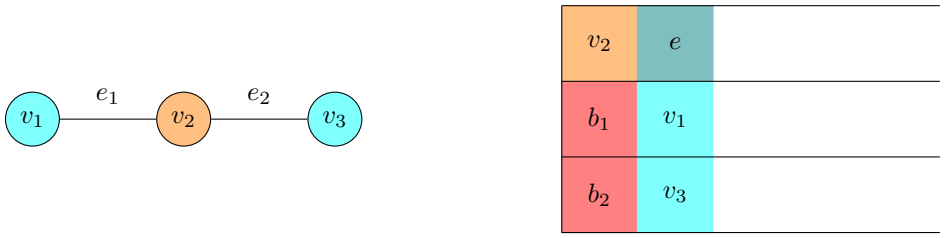


Figure 3 Reduction of Vertex Cover with $n' = 3$ and $k = 1$ to $P|and/or\text{-}prec, p_j = 1|\gamma$ with $k_s = 1$.

From the proof of Theorem 9, we moreover get the following corollary:

▶ **Corollary 10.** *$P|or/and\text{-}prec,\ p_j = 1|C_{\max}$ is $W[1]$-hard if parameterized by $C_{\max}$ and k_s.*

Given the complexity result in Theorem 9, the existence of a fixed-parameter tractable algorithm for $P|or/and\text{-}prec,\ p_j = 1|\gamma$ parameterized by k_s is unlikely. In the context of *and/or*-constraints, we provide even stronger evidence against the fixed-parameter tractability of $P|and/or\text{-}prec,\ p_j = 1|\gamma$ parameterized by k_s, by proving that the problem is para-$\mathcal{NP}$-hard.

▶ **Proposition 11.** *$P|and/or\text{-}prec,\ p_j = 1|\gamma$, $\gamma \in \{C_{\max}, \sum_j C_j\}$ is $\mathcal{NP}$-hard, even if $k_s = 1$.*

Proof. We prove the claim by a reduction from Vertex Cover. Let $I' = (G, k)$ be an instance of Vertex Cover, where $G = (V, E)$ is an undirected Graph and $k \in \mathbb{N}$ is the required size of a vertex cover. We may assume that $k < |V| =: n'$, otherwise I is trivial. We construct from I' an instance I of $P|and/or\text{-}prec,\ p_j = 1|\gamma$ with a single successor. We consider $m := n'$ machines. The set J of jobs is composed of a job v for each vertex $v \in V$, a single job e, which will be the only successor and $n' - k$ auxiliary jobs $b_1, ..., b_{n'-k}$. Through precedence constraints, we force all auxiliary jobs as well as one end-vertex per edge to be processed before e: We set

$$\psi_e = \bigwedge_{i=1}^{n'-k} x_{b_i} \wedge \bigwedge_{\{v,w\} \in E} (x_v \vee x_w)$$

and $\psi_j = \mathbb{1}$ for all $j \in J \setminus \{e\}$. We require to find a feasible schedule $\mathcal{S}$ with $C_{\max} \leq 2$ if $\gamma = C_{\max}$ or $\sum_{j \in J} C_j \leq 3n' - 2k + 2$ if $\gamma = \sum_j C_j$. It is easy to see that I' is a yes-instance precisely if we can find a schedule for I', which processes all auxiliary jobs and a vertex cover of size k during the first time slot and all remaining jobs during time slot two. The construction is visualized in Figure 3. ◀

From the proof of Proposition 11, we also get the following corollary:

▶ **Corollary 12.** *$P|and/or\text{-}prec,\ p_j = 1|C_{\max}$ is para-$\mathcal{NP}$-hard, if parameterized by $C_{\max}$ and k_s.*

5 Parameterization by the number of machines

A classical parameter in the study of parameterized complexity of scheduling problems, is the number of machines. In [4], Coffman and Graham prove that the $\mathcal{NP}$-hard problem $P|prec,\ p_j = 1|C_{\max}$ becomes polynomial when restricted to two machines. For an arbitrary but constant number of machines, the complexity is open. However, Bodlaender and

Fellows [2] prove that $P|prec,\ p_j = 1|C_{\max}$ is $W[2]$-hard when parameterized by the number of machines. In the following, we prove that $P2|and{+}or\text{-}prec,\ p_j = 1|\gamma$ is $\mathcal{NP}$-hard for $\gamma \in \{C_{\max}, \sum_j C_j, \sum_j w_j C_j\}$ and thus $P|and{+}or\text{-}prec,\ p_j = 1|\gamma$ is para-$\mathcal{NP}$-hard when parameterized by the number of machines. This result is also interesting in the context of the interplay between *and*- and *or*-constraints: On the one hand, it shows that the introduction of *or*-constraints turns the polynomial-time solvable $P2|prec,\ p_j = 1|C_{\max}$ into an $\mathcal{NP}$-hard problem. On the other hand, the result below shows that the introduction of a single *and*-job turns the polynomial-time solvable $P|or\text{-}prec,\ p_j = 1|C_{\max}$ into an $\mathcal{NP}$-hard problem. For an *and+or*-constrained instance with successors in $J_s = J_s^{or} \dot\cup J_s^{and}$, we denote by $k_s^{and} := |J_s^{and}|$ the number of *and*-successors.

▶ **Theorem 13.** *$P2|and{+}or\text{-}prec,\ p_j = 1|\gamma$, $\gamma \in \{C_{\max}, \sum_j C_j, \sum_j w_j C_j\}$ with $k_s^{and} = 1$ is $\mathcal{NP}$-hard.*

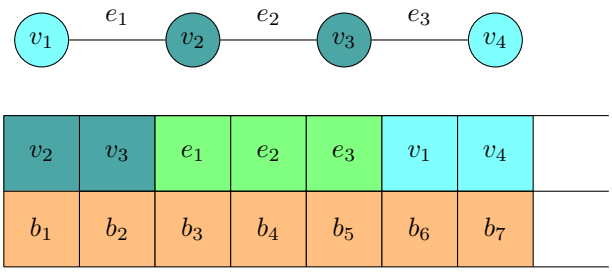


Figure 4 Reduction from Vertex Cover to $P2|and{+}or\text{-}prec, p_j = 1|\gamma$ with $J_s^{and} = \{b_6\}$.

Proof. We prove the claim by a reduction from Vertex Cover. Let $I = (G, k)$ be an instance of Vertex Cover, where $G = (V, E)$ is an undirected graph and k is the requested size of a vertex cover. We assume that $k < |V|$, otherwise I is trivial. We construct from I an instance I' of $P2|and{+}or\text{-}prec,\ p_j = 1|\gamma$, $\gamma \in \{C_{\max}, \sum_j C_j\}$, where the idea is the following: We introduce one job per edge, one job per vertex and $\ell := |V| + |E|$ auxiliary jobs to block the second machine. Using precedence constraints, we will force a feasible schedule to process all jobs in a vertex cover of size at most k before finishing the edge jobs.

More precisely, we set $J := V \cup E \cup B$ where $B = \{b_1, \dots, b_\ell\}$ is the set of auxiliary jobs. All vertex jobs, as well as the first auxiliary job, may start at any time, i.e. $\psi_j = \mathbb{1}$ if $j \in V \cup \{b_1\}$. Edges may not be scheduled unless one of their end-vertices has been scheduled already, i.e. $\psi_e = x_u \vee x_v$ for $e = \{u, v\} \in E$. We set

$$\psi_{b_{k+|E|+1}} = x_{b_{k+|E|}} \wedge \bigwedge_{e \in E} x_e$$

to make sure that all edge jobs have been scheduled by the time we start the $(k + |E| + 1)$-st auxiliary job. To process all auxiliary jobs in a chain, we set $\psi_{b_i} = x_{b_{i-1}}$, $i \in \{2, \dots, \ell\} \setminus \{k + |E| + 1\}$. Observe that $J_s^{and} := \{b_{k+|E|+1}\}$ and $J_s^{or} := E \cup \{b_2, \dots, b_{k+|E|}, b_{k+|E|+2}, \dots, b_\ell\}$ is a partition of the successors into *or*- and *and*-jobs. The construction is visualized in Figure 4. It is easy to see that I is a yes-instance, precisely if there exists a schedule for I' with $C_{\max} = \ell$, $(\sum_j C_j = \ell \cdot (\ell + 1))$. ◀

6 Summary and Outlook

In this paper, we study the problems of minimizing the makespan and the sum of completion times in parallel machine scheduling of unit-time jobs under precedence constraints. In our setting, each job j is associated with a precedence constraint ψ_j, encoded as a Boolean formula over the set of jobs. A schedule satisfies ψ_j if setting exactly the jobs scheduled before j to true satisfies the formula. Depending on the structure of these formulas, we distinguish between *and*-, *or*-, *and+or*-, *or/and*-, and *and/or*-constraints.

To analyze the parameterized complexity of these problems, we introduce two new structural parameters: the total number of predecessors and the total number of successors. Restricting the number of predecessors yields fixed-parameter tractability, even for general precedence constraints. In contrast, restricting the number of successors leads to a more nuanced landscape: for traditional *and*-constraints and more general *and+or*-constraints, the problem is FPT, but for *or/and*-constraints (DNF), it becomes $W[1]$-hard. For and/or-constraints (CNF), the problem is even para-$\mathcal{NP}$-hard. A common parameter in the analysis of scheduling problems is the number of machines. While minimizing the makespan for *and*-constrained unit jobs is known to be $W[2]$-hard parameterized by the number of machines (Bodlaender and Fellows [2]), we show that the same problem is para-$\mathcal{NP}$-hard in the presence of *and+or*-constraints.

We believe these results open up several promising research directions. One such direction concerns the interplay between *and*- and *or*-constraints. Naturally, any known hardness result for problems with classical *and*-constraints carries over to the more general version of the problem with *and+or*-constraints. Conversely, certain *or*-constrained problems can be solved in polynomial time, while they are $\mathcal{NP}$-hard in the *and*-constrained version (Berit [11]). However, Theorem 13 shows that it is inaccurate to think that the hardness of *and+or*-constrained problems stems only from the *and*-constraints: While minimizing the makespan or sum of completion times for unit jobs on two machines is in $\mathcal{P}$, both in the *or*- and the *and*-constrained version (Coffman and Graham [4], Berit [11]), we show that scheduling with *and+or*-constraints on just two machines is $\mathcal{NP}$-hard. We suggest studying parameters that measure the ratio between *or*- and *and*-constraints. In Theorem 13, we show that $P2|\textit{and+or-prec},\ p_j = 1|C_{\max}$ is para-$\mathcal{NP}$-hard parameterized by the number of *and*-successors, while Theorem 8 proves the same problem to be FPT parameterized by the total number of successors. Open questions in the same spirit include whether restricting only the predecessors of *and*-jobs or only the number of *or*-successors on two machines leads to tractability or retains hardness.

Table 1 Overview of parameterized complexity results for parallel machine scheduling of unit-time jobs with generalized precedence constraints.

Problem	γ	Parameter	Complexity	Source
$P\ \|\textit{gen-prec},\ p_j = 1\|\gamma$	$C_{\max},\ \sum_j C_j$	k_p	in $\mathcal{FPT}$	Thm 2
$P\ \|\textit{gen-prec},\ p_j = 1\|\gamma$	$\sum_j w_j C_j$	k_p	in $\mathcal{XP}$	Thm 2
$P\ \|\textit{and + or-prec},\ p_j = 1\|\gamma$	$C_{\max},\ \sum_j C_j$	k_s	in $\mathcal{FPT}$	Thm 8
$P2\|\textit{and + or-prec},\ p_j = 1\|\gamma$	$C_{\max},\ \sum_j C_j,\ \sum_j C_j$	–	$\mathcal{NP}$-hard	Thm 13
$P\ \|\textit{and + or-prec},\ p_j = 1\|\gamma$	$C_{\max},\ \sum_j C_j,\ \sum_j C_j$	$m,\ k_s^{and}$	para-$\mathcal{NP}$-hard	Thm 13
$P\ \|\textit{or/and-prec},\ p_j = 1\|\gamma$	$C_{\max},\ \sum_j C_j$	k_s	in $\mathcal{XP}$	Cor 5
$P\ \|\textit{or/and-prec},\ p_j = 1\|\gamma$	$C_{\max},\ \sum_j C_j,\ \sum_j w_j C_j$	k_s	$W[1]$-hard	Thm 9
$P\ \|\textit{and/or-prec},\ p_j = 1\|\gamma$	$C_{\max},\ \sum_j C_j$	k_s	para-$\mathcal{NP}$-hard	Prop 11

Moreover, our results show that it is fruitful to explore parameters that limit job dependencies, especially for problems that become hard due to precedence constraints. This perspective contrasts with the focus on the width in classical *and*-constrained problems, which instead promotes high interrelatedness among jobs. In *and*-constrained problems, parameters of the precedence graph, which reduce interrelatedness, may include the size of a largest clique or tournament and the largest out- or in-degree. In particular, we propose a hybrid approach: combining parameters that limit the portion of input influenced by precedence constraints with parameters that enforce high interrelatedness within affected jobs.

References

1 Stéphane Bessy and Rodolphe Giroudeau. Parameterized complexity of a coupled-task scheduling problem. *Journal of Scheduling*, 22(3):305–313, 2019. doi:10.1007/S10951-018-0581-1.

2 Hans L Bodlaender and Michael R Fellows. W[2]-hardness of precedence constrained k-processor scheduling. *Operations Research Letters*, 18(2):93–97, 1995. doi:10.1016/0167-6377(95)00031-9.

3 Lin Chen, Dániel Marx, Deshi Ye, and Guochuan Zhang. Parameterized and approximation results for scheduling with a low rank processing time matrix. In *34th Symposium on Theoretical Aspects of Computer Science (STACS 2017)*, pages 22–1. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.STACS.2017.22.

4 Edward G Coffman and Ronald L Graham. Optimal scheduling for two-processor systems. *Acta informatica*, 1:200–213, 1972. doi:10.1007/BF00288685.

5 Thomas Erlebach, Vanessa Kääb, and Rolf H Möhring. Scheduling and/or-networks on identical parallel machines. In *International workshop on approximation and online algorithms*, pages 123–136. Springer, 2003. doi:10.1007/978-3-540-24592-6_10.

6 Donald W Gillies and Jane W-S Liu. Scheduling tasks with and/or precedence constraints. *SIAM Journal on Computing*, 24(4):797–810, 1995. doi:10.1137/S0097539791218664.

7 Michael Goldwasser, J-C Latombe, and Rajeev Motwani. Complexity measures for assembly sequences. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 2, pages 1851–1857. IEEE, 1996.

8 Ronald L Graham. Bounds for certain multiprocessing anomalies. *Bell system technical journal*, 45(9):1563–1581, 1966.

9 Ronald Lewis Graham, Eugene Leighton Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of discrete mathematics*, volume 5, pages 287–326. Elsevier, 1979.

10 Felix Happach. Makespan minimization with or-precedence constraints. *Journal of Scheduling*, 24(3):319–328, 2021. doi:10.1007/S10951-021-00687-6.

11 Berit Johannes. On the complexity of scheduling unit-time jobs with or-precedence constraints. *Operations research letters*, 33(6):587–596, 2005. doi:10.1016/J.ORL.2004.11.009.

12 Dušan Knop and Martin Koutecký. Scheduling meets n-fold integer programming. *Journal of Scheduling*, 21:493–503, 2018. doi:10.1007/S10951-017-0550-0.

13 Sanghyup Lee, Ilkyeong Moon, Hyerim Bae, and Jion Kim. Flexible job-shop scheduling problems with "AND"/"OR" precedence constraints. *International Journal of Production Research*, 50(7):1979–2001, 2012.

14 Jan Karel Lenstra and AHG Rinnooy Kan. Complexity of scheduling under precedence constraints. *Operations Research*, 26(1):22–35, 1978. doi:10.1287/OPRE.26.1.22.

15 Matthias Mnich and René Van Bevern. Parameterized complexity of machine scheduling: 15 open problems. *Computers & Operations Research*, 100:254–261, 2018. doi:10.1016/J.COR.2018.07.020.

16 Matthias Mnich and Andreas Wiese. Scheduling and fixed-parameter tractability. *Mathematical Programming*, 154(1):533–562, 2015. doi:10.1007/S10107-014-0830-9.

17 Rolf H Möhring, Martin Skutella, and Frederik Stork. Scheduling with and/or precedence constraints. *SIAM Journal on Computing*, 33(2):393–415, 2004. doi:10.1137/S009753970037727X.

18 Jesper Nederlof and Céline M. F. Swennenhuis. On the Fine-Grained Parameterized Complexity of Partial Scheduling to Minimize the Makespan. In Yixin Cao and Marcin Pilipczuk, editors, *15th International Symposium on Parameterized and Exact Computation (IPEC 2020)*, volume 180 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:17, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.IPEC.2020.25.

19 Damien Prot and Odile Bellenguez-Morineau. A survey on how the structure of precedence constraints may change the complexity class of scheduling problems. *Journal of Scheduling*, 21(1):3–16, 2018. doi:10.1007/S10951-017-0519-Z.

20 Ravi Sethi. On the complexity of mean flow time scheduling. *Mathematics of Operations Research*, 2(4):320–330, 1977. doi:10.1287/MOOR.2.4.320.

21 René Van Bevern, Robert Bredereck, Laurent Bulteau, Christian Komusiewicz, Nimrod Talmon, and Gerhard J Woeginger. Precedence-constrained scheduling problems parameterized by partial order width. In *International conference on discrete optimization and operations research*, pages 105–120. Springer, 2016. doi:10.1007/978-3-319-44914-2_9.

Binary k-Center with Missing Entries: Structure Leads to Tractability

Tobias Friedrich
Hasso Plattner Institute (HPI), University of Potsdam, Germany

Kirill Simonov
Department of Informatics, University of Bergen, Norway

Farehe Soheil
Hasso Plattner Institute (HPI), University of Potsdam, Germany

Abstract

k-Center clustering is a fundamental classification problem, where the task is to categorize the given collection of entities into k clusters and come up with a representative for each cluster, so that the maximum distance between an entity and its representative is minimized. In this work, we focus on the setting where the entities are represented by binary vectors with missing entries, which model incomplete categorical data. This version of the problem has wide applications, from predictive analytics to bioinformatics.

Our main finding is that the problem, which is notoriously hard from the classical complexity viewpoint, becomes tractable as soon as the known entries are sparse and exhibit a certain structure. Formally, we show fixed-parameter tractable algorithms for the parameters vertex cover, fracture number, and treewidth of the row-column graph, which encodes the positions of the known entries of the matrix. Additionally, we tie the complexity of the 1-cluster variant of the problem, which is famous under the name Closest String, to the complexity of solving integer linear programs with few constraints. This implies, in particular, that improving upon the running times of our algorithms would lead to more efficient algorithms for integer linear programming in general.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases Clustering, Missing Entries, k-Center, Parameterized Algorithms

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.8

Related Version *Full Version*: `https://arxiv.org/abs/2503.01445`

Funding *Farehe Soheil*: Supported by the HPI Research School on Foundations of AI (FAI).

Acknowledgements We thank Nikolai Karpov for pointing out the applications in bioinformatics.

1 Introduction

Clustering is a fundamental problem in computer science with a wide range of applications [28, 30, 49, 25, 51], and has been thoroughly explored [39, 3, 31, 9, 52, 4]. In its most general formulation, given n data points, the aim of a clustering algorithm is to partition these points into groups, called clusters, based on the similarity. The degree of similarity between points is modeled by a given distance function. Depending on the representation of the data points, several computationally different variants of the clustering problem arise. Commonly, the points are embedded in the d-dimensional Euclidean space $\mathbb{R}^d$ with the standard Euclidean distance or the distance given by L_p norms, or in the space of binary strings equipped with the Hamming distance, or in a general metric space with an explicit given distance function.

Additionally, there exist different approaches to how the similarity is aggregated. For the purpose of this work, we focus on the classical center-based clustering objectives. In k-Center clustering, given a set of data points and the parameter k, the objective is to partition the points into k clusters and identify for each cluster a point called the *center*, so

© Tobias Friedrich, Kirill Simonov, and Farehe Soheil;
licensed under Creative Commons License CC-BY 4.0

that the *maximum* distance between the center and any point within its cluster is minimized. k-Median clustering is defined in the same way, except that the *sum* of distances between the data points and their respective centers is minimized, and k-Means clustering aims to minimize the sum of *squared* distances instead.

Unfortunately, virtually all versions of clustering are computationally hard, in the classical sense. k-Means in $\mathbb{R}^d$ is NP-hard even on the plane (dimension $d = 2$) [42], and it is also NP-hard for $k = 2$ clusters even when the vectors have binary entries [2, 18]. k-Median and k-Center are NP-hard in $\mathbb{R}^2$ as well [44]. Moreover, k-Center on binary strings under the Hamming distance is NP-hard even for $k = 1$ cluster; that is, the problem of finding the binary string that minimizes the maximum distance to a given collection of strings is already NP-hard [19, 37]. The latter problem is well-studied in the literature under the name of Closest String, given its important applications in coding theory [35] to bioinformatics [50].

In order to circumvent the general hardness results and simultaneously increase the modeling power of the problem, we consider the following variant of clustering with missing entries. We assume that the data points are represented by vectors, where each entry is either an element of the original domain (e.g., in $\mathbb{R}$ or $\{0, 1\}$), or the special element "?", which corresponds to an unknown entry. For the clustering objective, the distance is computed normally between the known entries, but the distance to an unknown entry is always zero. In this way, we can define the problems k-Center with Missing Entries and k-Means with Missing Entries. For formal definitions, we refer to Section 2.

These problems have a wide range of applications. For an example in predictive analytics, consider the setting of the well-known Netflix Prize challenge[1]. The input is a collection of user-movie ratings, and the task is to predict unknown ratings. The data can be represented in the matrix form, where the rows correspond to the users and the columns to the movies, and naturally *most* of the entries in this matrix would be unknown [46]. Clustering in this setting is then an important tool for grouping/labelling similar users or similar movies, based on the available data.

Clustering with missing entries is also closely related to string problems that arise in bioinformatics applications. In the fundamental genome phasing problem (known also as "haplotype assembly"), the input is a collection of *reads*, i.e., short subsequences of the two copies of the genome, and the task is to reconstruct both of the original sequences. Finding the best possible reconstruction in the presence of errors is then naturally modeled as an instance of k-Means with Missing Entries with $k = 2$, where the data points correspond to the individual reads, using missing entries to mark the unknown parts of each read; the target centers represent the desired complete genomic sequences; and the clustering objective represents the total number of errors between the known reads and the desired sequences, which needs to be minimized. In fact, [47] use exactly this formalization of the phasing problem (under the name of "Weighted Minimum Error Correction") as the algorithmic core of their WhatsHap phasing software. Note that in both examples above, the known entries lie in a finite, small domain. For the technical results in this work, we focus on vectors where the vectors have binary values, e.g., in $\{0, 1\}$; the results however can be easily extended to the bounded domain setting.

Generally speaking, k-Center with Missing Entries and k-Means with Missing Entries cannot be easier than their fully-defined counterparts. While greatly increasing the modeling power of the problem, the introduction of missing entries poses also additional

[1] The problem description and the dataset is available at `https://www.kaggle.com/netflix-inc/netflix-prize-data`.

technical challenges. In particular, most of the methods developed for the standard, full-information versions of clustering are no longer applicable, since the space formed by vectors with missing entries is not necessarily metric: the distances may violate triangle inequality. On the positive side, one can observe that in practical applications the structure of the missing entries is not completely arbitrary. In particular, the known entries are often *sparse* – for example, in the above-mentioned Netflix Prize challenge, only about 1% of the user-movie pairs have a known rating [46]. Therefore, the "hard" cases coming from the standard fully-defined versions of k-Center/k-Means are quite far from the instances arising in applications of clustering with missing entries. This motivates the aim to identify tractable cases of k-Center with Missing Entries based on the structure of the missing entries, since the general hardness results for k-Center are not applicable in this setting. Formally, we use the framework of *parameterized complexity* in order to characterize such cases. We refer to standard textbooks on parameterized complexity for a more thorough introduction to the subject [11, 10].

In order to apply the existing machinery and to put the parameters we consider into perspective, we encode the arrangement of the missing entries into a graph. We say that the *incidence graph* of a given instance is the following bipartite graph: the vertices are the data points and the coordinates, and the edge between a point and a coordinate is present when the respective entry is *known*. When interpreting the input as a matrix, where the data points are the rows, replacing the known entries by "1" and missing entries by "0" results exactly in the biadjacency matrix of the incidence graph. We call this matrix the *mask* matrix of the instance, denoted by $\boldsymbol{M}$, and denote the incidence graph of the instance by $G_{\boldsymbol{M}}$; see Figure 1 for an illustration.

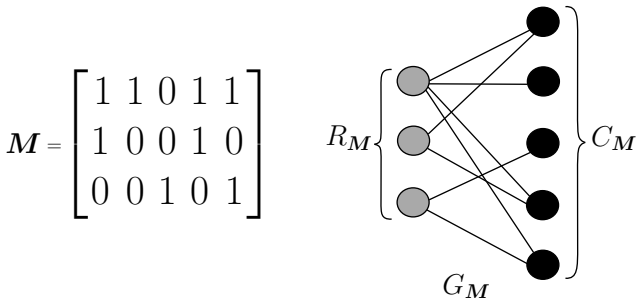


Figure 1 Mask matrix $\boldsymbol{M}$, and its corresponding incidence graph, on the right. Vertices corresponding to *rows* of $\boldsymbol{M}$, denoted by $R_{\boldsymbol{M}}$, are illustrated in gray. Vertices corresponding to *columns* of $\boldsymbol{M}$, denoted by $C_{\boldsymbol{M}}$, are in black.

We mainly consider the following three fundamental sparsity parameters of the incidence graph $G_{\boldsymbol{M}}$:

- Vertex cover number $\mathtt{vc}(G_{\boldsymbol{M}})$: the smallest number of vertices that are necessary to cover all edges of the graph;
- Fracture number $\mathtt{fr}(G_{\boldsymbol{M}})$: the smallest number of vertices one needs to remove so that the connected components of the remaining graph are small, i.e., their size is bounded by the same number;
- Treewidth $\mathtt{tw}(G_{\boldsymbol{M}})$: the classical decomposition parameter measuring how "tree-like" the graph is.

See Figures 2 and 3 for a visual representation of instances with small vertex cover and fracture number, respectively. Treewidth is the most general parameter out of the three, and it encompasses a wide range of instances. For example, the main algorithmic ingredient in the

WhatsHap genomic phasic software [47] is the FPT algorithm for k-Means with Missing Entries parameterized by the maximum number of known entries per column[2], in the special case where $k = 2$ and the known entries in each row form a continuous subinterval. In this setting, the treewidth of the incidence graph is never larger than this parameter. Vertex cover, on the other hand, is the most restrictive of the three; however, comparatively small vertex cover may be a feasible model in settings such as the Netflix Prize challenge, where most users would only have ratings for a relatively small collection of the most popular movies. The fracture number aims to generalize the setting of small vertex cover, to also allow for an arbitrary number of small local "information patches", outside of the few "most popular" rows and columns. Note that fracture number is a strictly more general parameter than vertex cover number, since removing any vertex cover from the graph results in connected components of size one; that is, for any instance, $\mathtt{fr}(G_M) \leq \mathtt{vc}(G_M)$. On the other hand, it holds that $\mathtt{tw}(G_M) \leq 2\,\mathtt{fr}(G_M)$, see Section 6 for the details.

Previously, the perspective outlined above has been successfully applied to k-Means with Missing Entries. Ganian et al. [20] show that the problem admits an FPT algorithm parameterized by treewidth of the incidence graph in case of the bounded domain, as well as further FPT algorithms for real-valued vectors in more restrictive parametrization. However, similar questions for k-Center with Missing Entries remain widely open.

In this work, we aim to close this gap and investigate parameterized algorithms for the k-Center objective on classes of instances where the known entries are "sparse", in the sense of the structural parameters above. Our motivation stems from the following. First, k-Center is a well-studied and widely applicable similarity objective, which in certain cases might be preferable over k-Means; specifically, whenever the cost of clustering is associated with each individual data point and has to be equally small, as opposed to minimizing the total, "social", cost spread out over all data points. Second, k-Center is interesting from a theoretical perspective, being a natural optimization target that on the technical level behaves very differently from k-Means. Our findings, as described next, show that the k-Center objective is in fact more challenging than k-Means in this context, and we prove that k-Center with Missing Entries is as general as a wide class of integer linear programs.

Our contribution. We present several novel parameterized algorithms for k-Center with Missing Entries. First, we show that k-Center with Missing Entries is FPT when parameterized by the vertex cover number $\mathtt{vc}(G_M)$ of the incidence graph plus k. Specifically, we prove the following result. Here and next, n is the number of data points in the instance and m is their dimension.

▶ **Theorem 1.** *k-Center with Missing Entries admits an algorithm with running time*

$$2^{\mathcal{O}\left(k\cdot\mathtt{vc}(G_M)+\mathtt{vc}(G_M)^2\cdot\log\mathtt{vc}(G_M)\right)}\,\mathrm{poly}(nm).$$

This result can be compared to the result of [15], who considered the complementary parametrization of the same problem (under the name of Any-Clustering-Completion). That is, they consider the minimum number of rows and columns that are needed to cover all *missing* entries. Interestingly, for their FPT algorithm, it was also necessary to include the target distance d in the parameter – we do not need this restriction in our setting, which highlights the property that instances of k-Center with Missing Entries, where missing entries are dense, are, in a sense, easier.

[2] Called *coverage* in their work.

Moving further, we extend the result of Theorem 1 to the more general setting where the parameter is the fracture number of the incidence graph $G_{\boldsymbol{M}}$. We show that one can achieve the running time of Theorem 1 even for fracture number, which is the most technical result of this paper.

▶ **Theorem 2.** *k-Center with Missing Entries admits an algorithm with running time*

$$2^{\mathcal{O}\left(k\cdot \mathtt{fr}(G_{\boldsymbol{M}})+\mathtt{fr}(G_{\boldsymbol{M}})^2\cdot\log \mathtt{fr}(G_{\boldsymbol{M}})\right)}\operatorname{poly}(nm).$$

To the best of our knowledge, no previous work on clustering problems considers the fracture number as the parameter; however it has been successfully applied to other fundamental problems such as Integer Linear Programs (ILPs) [24] and Edge Disjoint Paths [22]. Furthermore, a very similar parameter, equivalent to fracture number, has been studied in the literature under the name *vertex integrity*, for example in the context of Subgraph Isomorphism [5] and algorithmic metatheorems [36].

In order to prove Theorem 2, we also need an algorithm with respect to the treewidth of the incidence graph $G_{\boldsymbol{M}}$, stated in the next theorem. We denote by d the target radius of the cluster, i.e., the maximum distance between a point and its cluster center.

▶ **Theorem 3.** *k-Center with Missing Entries admits an algorithm with running time*

$$d^{\mathcal{O}\left(\mathtt{tw}(G_{\boldsymbol{M}})\right)}2^{\mathcal{O}\left(k\cdot\mathtt{tw}(G_{\boldsymbol{M}})\right)}\operatorname{poly}(nm).$$

In other words, the problem is FPT when parameterized by $\mathtt{tw}(G_m)+d+k$, or XP when parameterized by $\mathtt{tw}(G_m)+k$. Note that, as opposed to Theorem 1 and Theorem 2, here we need the dependence on d in the exponential part of the running time. This is, however, still sufficient to enable the algorithm claimed by Theorem 2.

While we are not aware of matching hardness results based on standard complexity assumptions, we can nevertheless argue that improving the running time in Theorems 1 and 2 resolves a fundamental open question. Specifically, we show a parameterized equivalence between Closest String, parameterized by the number of strings, and Integer Linear Program (ILP) with bounded variables, parameterized by the number of constraints. While we use the reduction from Closest String to ILP as a building block in the algorithm of Theorem 1, the reduction in the other direction, i.e., from ILP to Closest String, is most relevant here. Formally, it yields the following statement:

▶ **Theorem 4.** *For any $\alpha > 0$, assume that Closest String admits an algorithm with running time $2^{\mathcal{O}\left(n^{1+\alpha}\right)}\cdot\operatorname{poly}(n\ell)$, where n is the number of strings and ℓ is their length. Then the ILP $\{\boldsymbol{A}\boldsymbol{x}=\boldsymbol{b}:\forall i,\,\ell_i\le x_i\le u_i\}$, where $\boldsymbol{A}\in\mathbb{Z}^{r\times c}$, $\mathbf{b}\in\mathbb{Z}^r$, and $\ell_i\le u_i\in\mathbb{Z}$, can be solved in time $2^{\mathcal{O}\left(r^{1+\alpha+o(1)}\right)}\cdot\operatorname{poly}(rc)$, assuming that $\|\boldsymbol{A}\|_\infty=\mathcal{O}(r)$ and $\delta=2^{\mathcal{O}(r)}$, where $\delta=\max_{i\in[c]}(u_i-\ell_i)$.*

Note that Closest String is a special case of k-Center with Missing Entries with $k=1$ and no missing entries, where also the number of strings matches with the vertex cover of the known entries. Therefore, we immediately get that improving the $\mathtt{vc}(G_{\boldsymbol{M}})^2$ term in the exponent of Theorem 1 to $\mathtt{vc}(G_{\boldsymbol{M}})^{1+\alpha}$ implies the same improvement for this class of ILP instances, and the same holds for the result of Theorem 2. Notably, the result of Theorem 4 was independently discovered by Rohwedder and Węgrzycki in a recent paper [48]; they also provide further examples of problems, where such an improvement implies breaking long-standing barriers in terms of the best-known running time, and conjecture that this might be impossible.

Related work. Clustering problems are also extensively studied from the approximation perspective. The k-Center problem classically admits 2-approximation in poly-time [26, 17]. On the other hand, approximating k-Center in $\mathbb{R}^2$ within a factor of 1.82 is NP-hard [45, 7]. Katsikarelis et al. [32] study the (k, r)-center problem, where the task is to select at most k vertices (centers) in an edge-weighted graph, so that all the other vertices are at distance at most r from a center. They establish tight complexity bounds with respect to standard structural graph parameters (clique-width, vertex cover, tree-depth), including matching SETH/ETH lower bounds. Moreover they introduce parameterized approximation schemes that for any $\epsilon > 0$ efficiently compute a $(k, (1+\epsilon)r)$-center when parameterized by treewidth or clique-width of the graph.

Closest String is well-studied under various parameters [38, 27, 40, 8] and in the area of approximation algorithms [23, 38, 41, 43]. Abboud et al. [1] have shown that Closest String on binary strings of length ℓ can not be solved in time $(2-\varepsilon)^{\ell} \cdot \text{poly}(n\ell)$ under the SETH, for any $\varepsilon > 0$. This can be seen as a tighter analogue of Theorem 4 for the parametrization of Closest String by the lengths of the strings. A version of the Closest String with wildcards was studied from the viewpoint of parameterized complexity [29]; the wildcards behave exactly like missing entries in our definition, therefore this problem is equivalent to k-Center with Missing Entries for $k = 1$.

Knop et al. [33] studied combinatorial n-fold integer programming, yielding in particular $n^{\mathcal{O}(n^2)}\,\text{poly}(m)$ algorithms for Closest String, Closest String with wildcards, and k-Center (with binary entries), where n is the number of strings/rows and m is the length of the strings/number of coordinates. These results can be seen as special cases of our Theorem 2.

The versions of k-Center with Missing Entries/k-Means with Missing Entries with zero target cost of clustering is considered in the literature under the class of "matrix completion" problems. There, given a matrix with missing entries, the task is to complete it to achieve certain structure, such as few distinct rows (resulting in the same objective as in the clustering problems) or small rank. Parameterized algorithms for matrix completion problems were also studied in [21]. Moreover, Eiben et al. [14] study the Independent Set problem on induced subgraphs of powers of the hypercube, which corresponds to analyzing the "diversity" of a set of vectors, in contrast to minimizing the number of clusters. They also introduce the *Pow-Hyp-IS-Completion* variant, where given Boolean vectors with missing entries and integers k and r, the goal is to complete the vectors so that a subset of k vectors has pairwise Hamming distance at least $r + 1$, or determine that no such subset exists.

Finally, k-Means with Missing Entries has been studied in $\mathbb{R}^d$ from a parameterized approximation perspective. Geometrically, each data point with missing entries can be seen as an axis-parallel linear subspace of $\mathbb{R}^d$, and the task is to identify k centers as points in $\mathbb{R}^d$ that minimize the total squared distance to the assigned subspaces. [13] show that this problem admits a $(1+\varepsilon)$ approximation in time $2^{\text{poly}(k,\varepsilon,\Delta)} \cdot \text{poly}(nd)$, where Δ is the maximum number of missing entries per row, meaning that the subspaces have dimension at most Δ.

Paper organization. We define the necessary preliminaries in Section 2. Then we show the reduction from ILP to Closest String in Section 3. Sections 4–6 are dedicated to the respective FPT results for the parameters vertex cover, treewidth and fracture number. We provide a conclusion in Section 7. Due to space constraints, most of the technical proofs are deferred to the full version.

2 Preliminaries

In this section, we introduce key definitions and notations used throughout the paper. For an integer n we write $[n]$ to denote the set $\{1, \dots, n\}$. We use $\mathbb{Z}_+$ to denote the set of non-negative integers. For a vector of real numbers $\boldsymbol{v} \in \mathbb{R}^\ell$ with length ℓ, its i-th entry is denoted by $\boldsymbol{v}[i]$ and v_i interchangeably. By $\boldsymbol{v}^T$, we denote the transpose of the vector. Similarly, for a binary string $s \in \{0,1\}^\ell$ of length ℓ, its i-th bit is denoted by $s[i]$, and s_i and we write $s = [s_1, \dots, s_\ell]$ to denote the complete string. For a set $I \subseteq [\ell]$, the substring of s restricted to the indices in I is written as $s[I] \in \{0,1\}^{|I|}$. For two indices $i, j \in [\ell]$, the substring from $s[i]$ throughout $s[j]$ (inclusive) is denoted by $s[i,j] \in \{0,1\}^{j-i+1}$. The complement of s is represented by $\overline{s}$. The Hamming distance between two binary strings $s_1, s_2 \in \{0,1\}^\ell$ is denoted by $d_H : \{0,1\}^\ell \times \{0,1\}^\ell \to \mathbb{Z}_+$ with $d_H(s_1, s_2) = \big|\{i \in [\ell] : s_1[i] \neq s_2[i]\}\big|$. For a graph G, the set of vertices and edges are denoted by $V(G)$ and $E(G)$ respectively.
Consider a matrix $\boldsymbol{A}$ with n rows and m coordinates. The sets of rows and coordinates of $\boldsymbol{A}$ are denoted by $R_{\boldsymbol{A}} = \{r_1, r_2, \dots, r_n\}$ and $C_{\boldsymbol{A}} = \{c_1, c_2, \dots, c_m\}$ respectively. For $i \in [n]$ and $j \in [m]$, the i-th row of $\boldsymbol{A}$ is denoted by $\boldsymbol{A}[i]$ and the j-th entry of $\boldsymbol{A}[i]$ is written as $\boldsymbol{A}[i][j]$. We may also use a_i and a_{ij} respectively to refer to $\boldsymbol{A}[i]$ and $\boldsymbol{A}[i][j]$, when clear from context. For matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, with the same number of rows and columns, their *entry-wise* subtraction is written as $\boldsymbol{A} - \boldsymbol{B}$, meaning $(\boldsymbol{A} - \boldsymbol{B})[i][j] = \boldsymbol{A}[i][j] - \boldsymbol{B}[i][j]$. Similarly, their *entry-wise* product is denoted by $\boldsymbol{A} \circ \boldsymbol{B}$, where $(\boldsymbol{A} \circ \boldsymbol{B})[i][j] = \boldsymbol{A}[i][j] \cdot \boldsymbol{B}[i][j]$. For two binary matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, we define their row-wise Hamming distance as a column vector $d_H(\boldsymbol{A}, \boldsymbol{B})$ of length n, where, $d_H(\boldsymbol{A}, \boldsymbol{B})[i] = d_H(\boldsymbol{A}[i], \boldsymbol{B}[i])$. For two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ of the same length, $\boldsymbol{a} \leq \boldsymbol{b}$ if for all coordinate i, $\boldsymbol{a}[i] \leq \boldsymbol{b}[i]$.

Moreover, a missing entry is denoted by "?". We define the Hamming distance between a known entry (0 or 1) and an unknown entry (?) as zero, i.e., $d_H(0,?) = d_H(1,?) = 0$. We remark here that assuming the distance to a missing entry to be any other constant (e.g., 1 instead of 0) does not introduce additional complexity, as the number of missing entries in any given row is fixed by the input and their contribution to the cost of clustering does not depend on the solution. It is easy to observe that all the algorithms and techniques presented here would still apply in a straightforward manner to the problem with arbitrary cost of a missing entry, by adjusting the target radius of each row. For clarity of the exposition and consistency with existing literature on missing entries, we define the cost of a missing entry to be zero, as above.

For a binary matrix $\boldsymbol{M} \in \{0,1\}^{n \times m}$, its *incidence graph* is an undirected bipartite graph defined as $G_{\boldsymbol{M}} = (V_{\boldsymbol{M}}, E_{\boldsymbol{M}})$, where $V_{\boldsymbol{M}} = (R_{\boldsymbol{M}} \cup C_{\boldsymbol{M}})$ and $E_{\boldsymbol{M}} = \big\{(u,v) : u \in R_{\boldsymbol{M}}, v \in C_{\boldsymbol{M}}, \boldsymbol{M}[u][v] = 1\big\}$. We use *row* (resp. *coordinate*) vertices to represent the vertices in $G_{\boldsymbol{M}}$ that correspond to the rows (coordinates) of $\boldsymbol{M}$. For a visual representation of the incidence graph, refer to Figure 1.

With these definitions, we can now proceed to formalize the main problem in question. Note that we assume the input matrix $\boldsymbol{A}$ to only contain $\{0,1\}$ entries, as the missing entries are encoded by the matrix $\boldsymbol{M}$, and when $\boldsymbol{M}[i][j] = 0$, its product with $\boldsymbol{A}[i][j]$ is always 0.

▶ **Definition 5** (k-Center with Missing Entries). *Given matrices $\boldsymbol{A} \in \{0,1\}^{n \times m}$, and $\boldsymbol{M} \in \{0,1\}^{n \times m}$, integers k and d. The task is to determine if there exists a binary matrix $\boldsymbol{B} \in \{0,1\}^{n \times m}$ with at most k distinct rows such that:*

$$d_H\big((\boldsymbol{M} \circ \boldsymbol{A}), (\boldsymbol{M} \circ \boldsymbol{B})\big) \leq \boldsymbol{d},$$

where $\boldsymbol{d}$ is the n-dimensional column vector $(d, d, \dots, d)$.

Note that in the context of a k-Center with Missing Entries instance, we use the term "point" interchangeably with "row" and "coordinate" interchangeably with "column", since the points to be clustered are represented as rows of the matrices $\boldsymbol{A}$ and $\boldsymbol{M}$. We also say that an entry of $\boldsymbol{A}$ is a missing entry or "?" whenever the corresponding entry of $\boldsymbol{M}$ is zero.

Next, we define the related problems.

▶ **Definition 6** (Closest String). *Given a set of $n \in \mathbb{Z}_+$ binary strings $S = \{s_1, \ldots, s_n\}$ each of length ℓ and a non-negative integer $d \in \mathbb{Z}_+$, determine whether there exist a binary string s^* of length ℓ such that for all $i \in [n]$:*

$$d_H(s^*, s_i) \leq d.$$

▶ **Definition 7** (Non-uniform Closest String). *Given a set of $n \in \mathbb{Z}_+$ binary strings $S = \{s_1, \ldots, s_n\}$ each of length ℓ and a distance vector $\boldsymbol{d} = (d_1, \ldots, d_n) \in \mathbb{Z}_+^n$, determine whether there exist a binary string s^* of length ℓ such that for all $i \in [n]$,*

$$d_H(s^*, s_i) \leq d_i.$$

▶ **Definition 8** (ILP Feasibility). *Given a constraint matrix $\boldsymbol{A} \in \mathbb{Z}^{r \times c}$, a column vector $\mathbf{b} \in \mathbb{Z}^r$, and integers $\ell_i \leq u_i$ for each $i \in [c]$, determine whether there exists a column vector $\boldsymbol{x} \in \mathbb{Z}^c$ such that*

$$\boldsymbol{A} \cdot \boldsymbol{x} = \boldsymbol{b}$$

and $\ell_i \leq x_i \leq u_i$ for each $i \in [c]$.

Finally, we recall the known algorithms for Closest String/ILP Feasibility with respect to the number of strings/number of constraints. Knop et al. [33] designed a general approach to solve ILP formulations of various problems, including Closest String and Non-uniform Closest String, which is also applicable to versions with missing entries.

▶ **Theorem 9** (Knop et al. [33]). *An instance of Non-uniform Closest String with n binary strings of length ℓ can be solved in time $n^{\mathcal{O}(n^2)} \cdot \ell$. The running time also holds if the strings have missing entries.*

For ILP Feasibility where the coefficients are bounded, Eisenbrand and Weismantel [16] show a fast dynamic programming algorithm that has a similar running time.

▶ **Theorem 10** (Eisenbrand and Weismantel [16]). *An ILP of form $\max\{\boldsymbol{c}^T \boldsymbol{x} : \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, 0 \leq \boldsymbol{x} \leq \boldsymbol{u}, \boldsymbol{x} \in \mathbb{Z}^n\}$, where $\boldsymbol{A} \in \mathbb{Z}^{m \times n}$ with $\|\boldsymbol{A}\|_\infty \leq \Delta$, $\boldsymbol{b} \in \mathbb{Z}^m$, $\boldsymbol{u} \in \mathbb{Z}_+^n$, and $\boldsymbol{c} \in \mathbb{Z}^n$, can be solved in time $n \cdot \mathcal{O}(m)^{(m+1)^2} \cdot \mathcal{O}(\Delta)^{m \cdot (m+1)} \cdot \log^2 m \cdot \Delta$.*

3 Reduction from ILP Feasibility to Closest String

This section is dedicated to the reduction from ILP Feasibility to Closest String that preserves the number of rows up to a factor of $\mathcal{O}(\log \|\boldsymbol{A}\|_\infty)$, where $\boldsymbol{A}$ is the constraint coefficient matrix in the ILP Feasibility instance. We show the sequence of reductions that lead to the proof of Theorem 11, which is then used to establish Theorem 4. While complete proofs of Lemma 12 to Lemma 18 are provided in the full version, we state the results formally next.

▶ **Theorem 11.** *Let $\{\boldsymbol{Ax} = \boldsymbol{b} : \forall i, \ell_i \leq x_i \leq u_i\}$ be an ILP Feasibility instance, with $\boldsymbol{A} \in \mathbb{Z}^{r\times c}$, $\boldsymbol{b} \in \mathbb{Z}^r$, and $\ell_i, u_i \in \mathbb{Z}$ for each $i \in [c]$. In polynomial time, one can construct an equivalent instance of Closest String with $n = \mathcal{O}\big(\log \|\boldsymbol{A}\|_\infty \cdot r\big)$, $\ell = \mathcal{O}\big(\|\boldsymbol{A}\|_\infty \cdot \log^2 \|\boldsymbol{A}\|_\infty \cdot r \cdot c \cdot \delta\big)$, and $d = \mathcal{O}\big(\|\boldsymbol{A}\|_\infty \cdot \log \|\boldsymbol{A}\|_\infty \cdot c \cdot \delta\big)$; here, $\delta = \max_{i\in[c]}(u_i - \ell_i)$.*

We start with a lemma showing that a general ILP Feasibility instance, where the coefficients may be negative, and variables may have negative lower bounds, can be transformed, in linear time with respect to the number of variables, into an equivalent instance with non-negative variable domains, i.e., of form $0 \leq y_i \leq u'_i$, non-negative coefficient matrix and non-negative right-hand side, while mostly preserving the number of constraints, variables and the magnitude of the entries, by a constant factor.

▶ **Lemma 12.** *Let $\{\boldsymbol{Ax} = \boldsymbol{b} : \forall i, \ell_i \leq x_i \leq u_i\}$ be an ILP Feasibility instance with $\boldsymbol{A} \in \mathbb{Z}^{r\times c}$, and $\boldsymbol{b} \in \mathbb{Z}^r$, an integer column vector. In time $\mathcal{O}\big(r \cdot (r + c)\big)$, we can construct an equivalent instance of ILP Feasibility $\{\boldsymbol{A'y} = \boldsymbol{b'} : \forall i, 0 \leq y_i \leq u'_i\}$, with $\boldsymbol{b'} \in \mathbb{Z}_+^{2r}$, $\boldsymbol{A'} \in \mathbb{Z}_+^{2r\times(r+c)}$ such that $\|\boldsymbol{A'}\|_\infty = \|\boldsymbol{A}\|_\infty$, $\|\boldsymbol{b'}\|_\infty = \mathcal{O}\big(c\cdot\delta\cdot\|\boldsymbol{A}\|_\infty\big)$ and $\max\{u'_i\} = \mathcal{O}\big(c\cdot\delta\cdot\|\boldsymbol{A}\|_\infty\big)$, where $\delta = \max_{i\in[c]}(u_i - \ell_i)$.*

By Lemma 12, we observe that a general ILP Feasibility instance is equivalent to an instance where everything is non-negative, with a bounded blow-up in the number of constraints, variables and the magnitude of values. Thus, we continue our chain of reductions from a non-negative ILP. In the next step, we reduce from arbitrary bounds on the variables to binary variables, as stated in the following lemma.

▶ **Lemma 13.** *Let $\{\boldsymbol{Ax} = \boldsymbol{b} : \forall i, 0 \leq x_i \leq u_i\}$ be an ILP Feasibility instance with $\boldsymbol{A} \in \mathbb{Z}_+^{r\times c}$, a non-negative integer matrix with r rows and c columns and $\boldsymbol{b} \in \mathbb{Z}_+^r$, a column vector. In polynomial time, this instance can be reduced to an equivalent instance $\{\boldsymbol{A'y} = \boldsymbol{b'} : \forall i, y_i \in \{0,1\}\}$ of ILP Feasibility where $\boldsymbol{A'} \in \mathbb{Z}_+^{r\times c'}$ is a matrix with r rows, $c' \leq c \cdot \max\{u_i + 1\}$ columns and $\|\boldsymbol{A'}\|_\infty = \|\boldsymbol{A}\|_\infty$, and $\boldsymbol{b'} \in \mathbb{Z}_+^r$ is a vector with $\|\boldsymbol{b'}\|_\infty \leq \|\boldsymbol{b}\|_\infty$.*

Next, we show that an instance of ILP Feasibility with non-negative coefficients and binary variables can be reduced to an instance with binary coefficients. The proof is similar in spirit to the proof of Lemma 8 in [34], however we are interested in ILP Feasibility instances where variables are only allowed to take values in $\{0, 1\}$, which is the main technical difference.

▶ **Lemma 14.** *Let $\big\{\boldsymbol{Ax} = \boldsymbol{b} : \forall i, x_i \in \{0,1\}\big\}$ be an ILP Feasibility instance with $\boldsymbol{A} \in \mathbb{Z}_+^{r\times c}$, a non-negative integer matrix with r rows and c columns and $\boldsymbol{b} \in \mathbb{Z}_+^r$, a column vector. In polynomial time, this instance can be reduced to an equivalent instance $\big\{\boldsymbol{A'y} = \boldsymbol{b'} : \forall i, y_i \in \{0,1\}\big\}$ of ILP Feasibility where $\boldsymbol{A'}$ is a $\{0,1\}$-matrix with $r' = \mathcal{O}\big(\log \|\boldsymbol{A}\|_\infty \cdot r\big)$ rows and $c' = \mathcal{O}\big(\log \|\boldsymbol{A}\|_\infty \cdot (c + \|\boldsymbol{b}\|_\infty)\big)$ columns, where $\boldsymbol{b'} \in \mathbb{Z}_+^{r'}$ is a vector with $\|\boldsymbol{b'}\|_\infty = \mathcal{O}\big(\|\boldsymbol{b}\|_\infty\big)$.*

Now we introduce a lemma that allows to reduce a $\{0, 1\}$-coefficient matrix to a matrix with values only in $\{-1, 1\}$.

▶ **Lemma 15.** *Let $\{\boldsymbol{Ax} = \boldsymbol{b} : x_i \in \{0,1\}\}$ be an ILP Feasibility instance with $\boldsymbol{A} \in \{0,1\}^{r\times c}$ and $\boldsymbol{b} \in \mathbb{Z}_+^r$. In polynomial time, this instance can be reduced to an equivalent instance $\{\boldsymbol{A'y} = \boldsymbol{b'} : \forall i, y_i \in \{0,1\}\}$ of ILP Feasibility where $\boldsymbol{A'}$ is a $\{-1,1\}$-matrix with $r' = r+1$ rows and $c' = 2c$ columns and $\boldsymbol{b'} \in \mathbb{Z}_+^{r'}$ is a vector with $\|\boldsymbol{b'}\|_\infty = 2\|\boldsymbol{b}\|_\infty$.*

It will be more convenient to work with ILP instances of form $\boldsymbol{Ax} \leq \boldsymbol{b}$, which motivates the following lemma.

▶ **Lemma 16.** *Every ILP* Feasibility *instance* $\{\boldsymbol{Ax} = \boldsymbol{b} : \forall i,\, x_i \in \{0,1\}\}$ *with* $\boldsymbol{A} \in \{-1,+1\}^{r\times c}$ *and* $\boldsymbol{b} \in \mathbb{Z}^r_+$ *can be reduced to an equivalent ILP instance* $\{\boldsymbol{A'y} \leq \boldsymbol{b'} : \forall i,\, y_i \in \{0,1\}\}$ *with* $\boldsymbol{A} \in \{-1,+1\}^{2r\times c}$ *and* $\boldsymbol{b'} \in \mathbb{Z}^{2r}$*, where* $\|\boldsymbol{b'}\|_\infty = \|\boldsymbol{b}\|_\infty$.

Before reducing to Closest String itself, we first show a reduction to a slightly more general version of the problem, Non-uniform Closest String (Definition 7), where each string has its own upper bound on the distance to the closest string.

▶ **Lemma 17.** *Every ILP instance* $\{\boldsymbol{Ax} \leq \boldsymbol{b} : \forall i,\, x_i \in \{0,1\}\}$ *with* $\boldsymbol{A} \in \{-1,+1\}^{r\times c}$ *and* $\boldsymbol{b} \in \mathbb{Z}^r$ *can be reduced, in polynomial time, to a* Non-uniform Closest String *instance with* r *strings of length* c*, where* $d_i \leq \big(c + \|\boldsymbol{b}\|_\infty\big)$ *for each* $i \in [n]$.

Finally, we show a reduction from Non-uniform Closest String to Closest String.

▶ **Lemma 18.** *Every* Non-uniform Closest String *instance with strings* $S = \{s_1, \ldots, s_n\}$ *of length* $\ell \in \mathbb{N}$ *and distance vector* $\boldsymbol{d} = (d_1, \ldots, d_n)$ *can be reduced to a* Closest String *instance with* $2n$ *strings* $S' = \{s'_1, \ldots, s'_{2n}\}$ *and distance* $d = \max_{i=1}^n d_i$ *in time* $\mathcal{O}\big(n \cdot \ell\big)$*, where length of the constructed strings is at most* $\ell + 2n \cdot d$.

With the reductions above, we can conclude with the proof of Theorem 11 as outlined below.

Proof of Theorem 11. We start by applying Lemma 12, to obtain an equivalent ILP Feasibility instance where the coefficients, constraints and variable domains are all non-negative integers. Next, by applying Lemma 13, we get an equivalent instance of ILP Feasibility with binary variables. Then, with the help of Lemma 14, we get that also the coefficients in the constraints are binary. After applying Lemma 15, we get that all coefficients are either -1 or $+1$, and additionally that the ILP instance is of the form $\boldsymbol{Ax} \leq \boldsymbol{b}$ by applying Lemma 16. Finally, we reduce to Non-uniform Closest String via Lemma 17, and then to Closest String via Lemma 18.

We now argue about the size of the constructed instance. The number of rows is increased by a factor of $\mathcal{O}\big(\log \|\boldsymbol{A}\|_\infty\big)$ by the reduction of Lemma 14, and by at most a constant factor in all other reductions. Therefore, $n = \mathcal{O}\big(\log \|\boldsymbol{A}\|_\infty \cdot r\big)$. The number of columns is increased by the original number of rows in Lemma 12, by a factor of δ in Lemma 13, where $\delta = \max u_i - \ell_i + 1$, and then $\mathcal{O}\big(c \cdot \delta \cdot \|\boldsymbol{A}\|_\infty\big)$ additional columns are added in Lemma 14, along with an additional factor of $\mathcal{O}\big(\log \|\boldsymbol{A}\|_\infty\big)$. Lemma 15 only increases the number of columns by at most a constant factor, and Lemmata 16, 17 do not change the number of columns. Finally, in Lemma 18, d is set to at most $\mathcal{O}\big(\|\boldsymbol{A}\|_\infty \cdot \log \|\boldsymbol{A}\|_\infty \cdot c \cdot \delta\big)$ and additional $2n \cdot d$ columns are attached, where $n = \mathcal{O}\big(\log \|\boldsymbol{A}\|_\infty \cdot r\big)$. Therefore, the final length of the strings ℓ is at most $\mathcal{O}\big(\|\boldsymbol{A}\|_\infty \cdot \log^2 \|\boldsymbol{A}\|_\infty \cdot r \cdot c \cdot \delta\big)$. ◀

As mentioned in the previous section, ILP Feasibility admits an algorithm with running time $(r\Delta)^{(r+1)^2} \cdot \mathrm{poly}(rc)$ [16], where $\Delta = \|\boldsymbol{A}\|_\infty$. On the other hand, a version of ILP Feasibility without upper bounds on the variables can be solved in time $\mathcal{O}\big(c \cdot (r\Delta)^{2r} \cdot \|\boldsymbol{b}\|_1^2\big)$ [16], and this is tight under the ETH [34]. This raises a natural question whether the gap in the running time between the two versions is necessary. As Closest String admits a straightforward reduction to ILP Feasibility, it also makes sense to ask whether running time better than $n^{\mathcal{O}\left(n^2\right)} \cdot \mathrm{poly}(n\ell)$ can be achieved for Closest String. From Theorem 11,

it follows that an improved Closest String algorithm would automatically improve the best-known running time for ILP Feasibility (with upper bounds). Formally, we prove Theorem 4, as stated in Section 1.

▶ **Theorem 4.** *For any $\alpha > 0$, assume that Closest String admits an algorithm with running time $2^{\mathcal{O}(n^{1+\alpha})} \cdot \text{poly}(n\ell)$, where n is the number of strings and ℓ is their length. Then the ILP $\{\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} : \forall i,\ \ell_i \le x_i \le u_i\}$, where $\boldsymbol{A} \in \mathbb{Z}^{r\times c}$, $\mathbf{b} \in \mathbb{Z}^r$, and $\ell_i \le u_i \in \mathbb{Z}$, can be solved in time $2^{\mathcal{O}(r^{1+\alpha+o(1)})} \cdot \text{poly}(rc)$, assuming that $\|\boldsymbol{A}\|_\infty = \mathcal{O}(r)$ and $\delta = 2^{\mathcal{O}(r)}$, where $\delta = \max_{i\in[c]}(u_i - \ell_i)$.*

Proof. We construct a Closest String instance following Theorem 11, and then by applying the assumed algorithm for Closest String on the obtained instance, we obtain an algorithm for the original ILP Feasibility with running time:

$$2^{\mathcal{O}(n^{1+\alpha})} \cdot \text{poly}(n\ell),$$

where $n = \mathcal{O}(\log \|\boldsymbol{A}\|_\infty \cdot r)$, $\ell = \mathcal{O}(\|A\|_\infty \cdot \log^2 \|A\|_\infty \cdot r \cdot c \cdot \delta)$. This can be expressed as

$$\begin{aligned} 2^{\mathcal{O}(n^{1+\alpha})}\,\text{poly}(n\ell) &= 2^{\mathcal{O}((\log\|\boldsymbol{A}\|_\infty \cdot r)^{1+\alpha})} \cdot (\|\boldsymbol{A}\|_\infty \cdot r \cdot c \cdot \delta)^{\mathcal{O}(1)} \\ &= 2^{\mathcal{O}(r^{1+\alpha+o(1)})} \cdot r^{\mathcal{O}(1)} \cdot c^{\mathcal{O}(1)} \cdot 2^{\mathcal{O}(r)} \\ &= 2^{\mathcal{O}((r^{1+\alpha+o(1)})+1)} \cdot \text{poly}(rc). \\ &= 2^{\mathcal{O}(r^{1+\alpha+o(1)})} \cdot \text{poly}(rc). \end{aligned}$$

◀

From Theorem 1 in [34], and the chain of reductions in the proof of Theorem 11, we can also get the following hardness result for Closest String, that at least the running time of $2^{\mathcal{O}(n\log n)} \cdot \text{poly}(n\ell)$ is necessary, similarly to ILP Feasibility with unbounded variables.

▶ **Corollary 19.** *Assuming ETH, there is no algorithm that solves Closest String in time $2^{o(n\log n)} \cdot \text{poly}(n\ell)$.*

4 Vertex Cover

In this section, we describe an FPT algorithm for k-Center with Missing Entries parameterized by the vertex cover number of the incidence graph, $G_{\boldsymbol{M}}$, derived from mask matrix $\boldsymbol{M}$. More formally, we prove the following:

▶ **Theorem 1.** *k-Center with Missing Entries admits an algorithm with running time*

$$2^{\mathcal{O}(k\cdot \text{vc}(G_{\boldsymbol{M}}) + \text{vc}(G_{\boldsymbol{M}})^2 \cdot \log \text{vc}(G_{\boldsymbol{M}}))}\,\text{poly}(nm).$$

Proof. Let S be a minimum vertex cover of the incidence graph $G_{\boldsymbol{M}}$. Define sets $R_S = (S \cap R_{\boldsymbol{M}})$ and $C_S = (S \cap C_{\boldsymbol{M}})$, to represent rows and coordinates of $\boldsymbol{M}$ that are in the vertex cover S. Respectively, define $\overline{C_S} = C_{\boldsymbol{M}} \setminus C_S$, and $\overline{R_S} = R_{\boldsymbol{M}} \setminus R_S$, as their complements. Note that, by definition of $G_{\boldsymbol{M}}$, for every $r \in \overline{R_S}$ and $c \in \overline{C_S}$, it follows that $\boldsymbol{M}[r][c] = 0$. To see this, assume there exist $r_0 \in \overline{R_S}$, and $c_0 \in \overline{C_S}$ such that $\boldsymbol{M}[r_0][c_0] = 1$. Then, the edge (r_0, c_0) exists in $G_{\boldsymbol{M}}$ that is not covered by S, as neither r_0 nor c_0 belong to S, which contradicts that S is a vertex cover of $G_{\boldsymbol{M}}$. Thus, the cost of clustering only depends on the rows and coordinates in S, i.e. $(R_S \cap C_S)$. We refer R_S, and $\overline{R_S}$ as *long rows*, and *short rows*

respectively, as the latter has known values only along the coordinates in C_S. See Figure 2 for an illustration of the instance structure; rows and coordinates may need to be reordered to reflect this structure, but this does not affect the problem's objective or the algorithm. We also define the following important mappings:

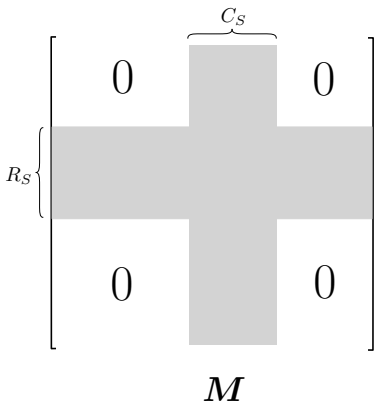


Figure 2 The mask matrix $\boldsymbol{M}$. The non-zero entries are distributed only within R_S and C_S. For every $r \in \overline{R_S}$ and $c \in \overline{C_S}$ it holds that $\boldsymbol{M}[r][c] = 0$.

- Partial cluster assignment; $\texttt{part} : R_S \to [k]$,
- Partial center assignment; $\texttt{cent} : [k] \times C_S \to \{0, 1\}$

Intuitively, `part` assigns each long row $r \in R_S$ to one of the k clusters, and `cent` assigns value to the centers of *all* clusters, along the coordinates in C_S. We call a pair $(\texttt{part}, \texttt{cent})$, a *partial assignment* and say it is *valid* if, $d_H\big((\boldsymbol{A} \circ \boldsymbol{M})[r][C_S], \texttt{cent}\big[\texttt{part}[r], C_S\big]\big) \leq d$, for every $r \in R_S$. Let P denote the set of all possible partial assignments. For a valid $(\texttt{part}, \texttt{cent}) \in P$, and cluster $j \in [k]$, define $T_{\texttt{part},j} = \{r \in R_S \mid \texttt{part}[r] = j\}$, as the set of *long* rows assigned to cluster j by `part`. We denote the cardinality of $T_{\texttt{part},j}$ by p_j.

With the necessary definitions set, we now proceed to outline Algorithm 1. Note that, in order to ensure a correct cluster assignment, we must ensure that each long row lies within distance d of its cluster center along coordinates $C_{\boldsymbol{M}}$, and similarly that each short row lies within distance d along C_S. Broadly, Algorithm 1 can be divided into the following key steps:

1. First, it establishes a valid partial assignment for the *long* rows along coordinates in C_S.
2. Then, for the clusters that have been assigned to the *long* rows, it tries to determine if there is a valid center assignment along the remaining coordinates, $\overline{C_S}$.
3. Finally, it decides whether a valid cluster assignment is also possible for the *short* rows.

We start by obtaining the minimum vertex cover of $G_{\boldsymbol{M}}$. Then we establish a valid partial assignment $(\texttt{part}, \texttt{cent})$ for the *long* rows along coordinates in C_S *(lines 1-4)*. Consequently, the remaining distance that each long row $r_i \in T_{\texttt{part},j}$ can have to its respective cluster center along $\overline{C_S}$ is bounded by $d'_i = d - d_H\big((\boldsymbol{A} \circ \boldsymbol{M})[r_i][C_S], \texttt{cent}[j, C_S]\big)$ *(lines 5-12)*.

Next, for clusters that have been assigned to the *long* rows, we try to determine if there exists a valid center assignment along the remaining coordinates, $\overline{C_S}$. To do so, we solve a binary Non-uniform Closest String instance (denoted as *NUCS* on line 14) for each *non-empty* cluster $j \in [k]$, with the distance vector $\boldsymbol{d}'_j = (d'_1, \ldots, d'_{p_j})$, and the strings $r'_j = \boldsymbol{A}[r_i][\overline{C_S}]$, for $i \in [p_j]$ *(lines 13-16)*. At this stage, we have a valid cluster assignment for the long rows, and the final task is to assign the short rows to appropriate clusters.

Finally, we decide whether a valid cluster assignment is also possible for the *short* rows. So, we assign each $r \in \overline{R_S}$, to the smallest $j \in [k]$ such that $d_H\big((\boldsymbol{A} \circ \boldsymbol{M})[r][C_S], \texttt{cent}[j, C_S]\big) \leq d$ *(lines 17-24)*. If at this step, we can assign all the *short* rows ($\overline{R_S}$) to some cluster center defined by $\texttt{cent}$, then the given k-Center with Missing Entries instance is feasible *(lines 25-26)*. Otherwise, we repeat the process for another valid partial assignment.

For correctness: First assume there is a solution, we show that our algorithm correctly finds it. Let $\texttt{part}^* : R_{\boldsymbol{M}} \to [k]$ be the solution cluster assignment and $\texttt{cent}^* : [k] \times C_{\boldsymbol{M}} \to \{0, 1\}$ be the solution center assignment. Then for each cluster $i \in [k]$ and each long row $r \in R_S$ assigned to it, that is $\texttt{part}^*[r] = i$, it holds that

$$\begin{aligned} d_H((\boldsymbol{A} \circ \boldsymbol{M})[r], \texttt{cent}^*[i]) &= d_H\big((\boldsymbol{A} \circ \boldsymbol{M})[r][C_S], \texttt{cent}^*[i][C_S]\big) \\ &\quad + d_H\big((\boldsymbol{A} \circ \boldsymbol{M})[r][\overline{C_S}], \texttt{cent}^*[i][\overline{C_S}]\big) \leq d \end{aligned}$$

Since we consider all the possible valid partial assignments for long rows in our algorithm, $\texttt{cent}^*[\texttt{part}^*[r]][C_S]$ is captured by at least one of the partial assignments $(\texttt{part}, \texttt{cent}) \in P$. In other words, there is at least one partial assignments $(\texttt{part}, \texttt{cent}) \in P$ such that $\texttt{part} = \texttt{part}^*|_{R_S}$ and $\texttt{cent} = \texttt{cent}^*|_{C_S}$, where $|$ denotes restriction to a set. Thus, by definition, it holds that $d_H((\boldsymbol{A} \circ \boldsymbol{M})[r][\overline{C_S}], \texttt{cent}^*[\texttt{part}^*[r]][\overline{C_S}]) \leq d'_r$. Consequently, the Non-uniform Closest String instance for cluster i and the long rows assigned to it, will correctly output feasible. As a result, for the long rows, we correctly output that a k-cluster with radius at most d is feasible. For the short rows, note that they have known entries only along coordinates C_S, so for each short row $r \in R_S$ assigned to cluster $i \in [k]$, that is $\texttt{part}^*[r] = i$, we have:

$$\begin{aligned} d_H((\boldsymbol{A} \circ \boldsymbol{M})[r], \texttt{cent}^*[i]) &= d_H\big((\boldsymbol{A} \circ \boldsymbol{M})[r][C_S], \texttt{cent}^*[i, C_S]\big) \\ &= d_H\big((\boldsymbol{A} \circ \boldsymbol{M})[r][C_S], \texttt{cent}[i, C_S]\big) \leq d. \end{aligned}$$

Hence, r will at least be assigned to cluster i. Therefore if there is a solution to this instance of k-Center with Missing Entries, then our algorithm will successfully output feasible. The other direction can similarly be showed.

As of the running time, in the first step, we obtain the minimum vertex cover of $G_{\boldsymbol{M}}$ can be computed in time $\mathcal{O}\big(1.2738^{\texttt{vc}(G_{\boldsymbol{M}})} + (\texttt{vc}(G_{\boldsymbol{M}}) \cdot nm)\big)$ by the state-of-the-art algorithm of [6]. For the next steps, note that there are at most $k^{|R_S|} = k^{\mathcal{O}\left(\texttt{vc}(G_{\boldsymbol{M}})\right)}$ possible mappings for $\texttt{part}$ and $2^{(|C_S| \cdot |J|)} = 2^{\mathcal{O}\left(\texttt{vc}(G_{\boldsymbol{M}}) \cdot k\right)}$ for $\texttt{cent}$. The Non-uniform Closest String instances are solved via Theorem 9 in time $m \cdot \texttt{vc}(G_{\boldsymbol{M}})^{\mathcal{O}\left(\texttt{vc}(G_{\boldsymbol{M}})\right)}$. Note that because the distance of each short row $r \in \overline{R_S}$ to a cluster center can be checked in time $(k' \cdot |C_S|) = \mathcal{O}\big(k \cdot \texttt{vc}(G_{\boldsymbol{M}})\big)$, the final step can be computed in time $\mathcal{O}\big(m \cdot k \cdot \texttt{vc}(G_{\boldsymbol{M}}) \cdot 2^{(\texttt{vc}(G_{\boldsymbol{M}}) \cdot k)}\big)$. In total, the running time of the algorithm is dominated by $2^{\mathcal{O}\left(k \cdot \texttt{vc}(G_{\boldsymbol{M}}) + \texttt{vc}(G_{\boldsymbol{M}})^2 \cdot \log \texttt{vc}(G_{\boldsymbol{M}})\right)} \operatorname{poly}(nm)$. ◀

5 Treewidth

This section is dedicated to a fixed-parameter algorithm for the problem parameterized by the treewidth of the incidence graph $G_{\boldsymbol{M}}$, the number of clusters k and the maximum permissible radius of the cluster, d. We restate the formal result next for convenience.

▶ **Theorem 3.** *k-Center with Missing Entries admits an algorithm with running time*

$$d^{\mathcal{O}\left(\texttt{tw}(G_{\boldsymbol{M}})\right)} 2^{\mathcal{O}\left(k \cdot \texttt{tw}(G_{\boldsymbol{M}})\right)} \operatorname{poly}(nm).$$

Algorithm 1 k-Center with Missing Entries parameterized by $\mathtt{vc}(G_{\boldsymbol{M}})$.

```
1  S ← Minimum Vertex Cover of G_M
2  P ← {(part, cent) ∈ (R_S → [k]) × ([k] × C_S → {0,1})}
3  for (part, cent) ∈ P and isValid(part, cent) do
4      soln ← true
5      for j ∈ [k] do
6          T_j ← {r ∈ R_S | part[r] = j}
7          if T_j = ∅ then
8              continue
9          for r_i ∈ T_j do
10             d'_i = d − d_H((A ∘ M)[r_i][C_S], cent[j, C_S])
11             r'_i = A[r_i][C̄_S]
12         d' ← (d'_1, ..., d'_{p_j})
13         r' ← (r'_1, ..., r'_{p_j})
14         soln ← soln ∧ NUCS(k, d', r')
15         if !soln then
16             break
17     if soln then
18         counter ← 0
19         for r ∈ R̄_S do
20             for j ∈ [k] do
21                 if d_H((A ∘ M)[r][C_S], cent[j, C_S]) ≤ d then
22                     counter++
23                     break
24         if counter = |R̄_S| then
25             return YES
26 return NO
```

We briefly sketch the intuition of our approach. For a complete proof and discussion, refer to the full version. Informally, the fact that $G_{\boldsymbol{M}}$ has treewidth at most t means that the graph can be constructed in a tree-like fashion, where at each point only a vertex subset of size at most t is "active". This small subset is called a *bag*, and in particular is a separator for the graph: there are no edges between the "past", already constructed part of the graph and the "future", not encountered yet part of the graph; all connections between these parts are via the bag itself. In terms of the k-Center with Missing Entries instance, this means that for each "past" row, all its entries that correspond to "future" columns are missing (as the respective entry of the mask matrix $\boldsymbol{M}$ has to be 0), and the same holds for "past" columns and "future" rows.

Our algorithm performs dynamic programming over this decomposition, supporting a collection of records for the current bag. For the rows of the current bag, we store the partition of the rows into clusters, and additionally for each row the distance to its center vector among the already encountered columns. For each cluster and for each column of the bag, we store the value of the cluster center in this column. These three characteristics (called together a *fragment*) act as a "trace" of a potential solution on the current bag. In

our DP table, we store whether there exists a partial solution for each choice of the fragment. Because of the separation property explained above, only knowing the fragments is sufficient for computing the records for every possible update on the bag. The claimed running time follows from upper-bounding the number of possible fragments for a bag of size at most t.

6 Fracture number

In this section, we present a fixed-parameter algorithm for the k-Center with Missing Entries, where the parameter is the fracture number of the incidence graph $G_{\boldsymbol{M}}$. To provide context, we first define the key concepts of *fracture modulator* and *fracture number*. Following this, we exploit the relevant results from Theorem 1 and Theorem 3, which lead to the proof of Theorem 2.

▶ **Theorem 2.** *k-Center with Missing Entries admits an algorithm with running time*

$$2^{\mathcal{O}\left(k\cdot \mathtt{fr}(G_{\boldsymbol{M}})+\mathtt{fr}(G_{\boldsymbol{M}})^2\cdot\log \mathtt{fr}(G_{\boldsymbol{M}})\right)}\operatorname{poly}(nm).$$

For a graph $G=(V,E)$, a *fracture modulator* is a subset of vertices $F\subseteq V$ such that, after removing the vertices in F, each remaining connected component contains at most $|F|$ vertices; the size of the smallest fracture modulator is denoted by $\mathtt{fr}(G)$. Consider the incidence graph $G_{\boldsymbol{M}}$ corresponding to the mask matrix $\boldsymbol{M}$, and let F represent the fracture modulator of $G_{\boldsymbol{M}}$ with the smallest cardinality. We note that F can be computed in time $\mathcal{O}\big((\mathtt{fr}(G)+1)^{\mathtt{fr}(G)}nm\big)$ by the algorithm of [12]. Let $R_F=(R_{\boldsymbol{M}}\cap F)$ and $C_F=(C_{\boldsymbol{M}}\cap F)$ represent the row and column vertices of $G_{\boldsymbol{M}}$ that belong to the fracture modulator F, respectively. Additionally, define $\overline{R_F}=R_{\boldsymbol{M}}\setminus R_F$ and $\overline{C_F}=C_{\boldsymbol{M}}\setminus C_F$. We refer to the rows in R_F as *long* rows while the rows corresponding to $\overline{R_F}$ are called *short* rows. See Figure 3 for an illustration of the structure of the instance. Now, the algorithm considers two cases.

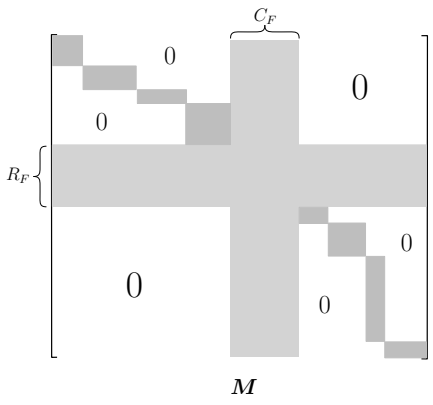


Figure 3 The rows R_F and columns C_F of the fracture modulator are in light gray, the "blocks" induced by the connected components of $G_{\boldsymbol{M}}\setminus F$ are in dark gray. All entries of $\boldsymbol{M}$ outside of the gray areas are 0.

$2\,\mathtt{fr}(G_{\boldsymbol{M}})\leq d$. In this case, each of the short rows has distance at most d to any fixed vector. This holds since for each row $i\in\overline{R_F}$, $\boldsymbol{M}[i][j]=1$ only for $j\in C_F$ or j in the same connected component of $G_{\boldsymbol{M}}\setminus F$ as i; this is at most $2\,\mathtt{fr}(G_{\boldsymbol{M}})\leq d$ entries, and all other entries in the row are missing. Therefore, the short rows are essentially irrelevant for the solution, as they can be assigned to any cluster with any center vector. We run the algorithm

of Theorem 1 on the instance restricted to the long rows R_F, and complement the resulting solution with an arbitrary assignment of the short rows to the k clusters. Since the vertex cover of the restricted instance is at most $|R_F| \leq \mathtt{fr}(G_M)$, the running time bound holds as desired.

$2\,\mathtt{fr}(G_M) > d$. Here, we use the algorithm of Theorem 3 to solve the given instance. We observe that a tree decomposition of width at most $2\,\mathtt{fr}(G_M)$ can be constructed for G_M in a straightforward fashion: create a bag for each connected component of $G_M \setminus F$ containing all vertices of this component together with F, and arrange these bags on a path in arbitrary order. Since $d < 2\,\mathtt{fr}(G_M)$, the running time of Theorem 3 gives the desired bound.

7 Conclusion

We have investigated the algorithmic complexity of k-Center with Missing Entries in the setting where the missing entries are sparse and exhibit certain graph-theoretic structure. We have shown that the problem is FPT when parameterized by $\mathtt{vc}+k$, $\mathtt{fr}+k$ and $\mathtt{tw}+k+d$, where $\mathtt{vc}$ is the vertex cover number, $\mathtt{fr}$ is the fracture number, and $\mathtt{tw}$ is the treewidth of the incidence graph. In fact, it is not hard to get rid of k in the parameter; for example, in the enumeration of partial center assignments in the algorithm of Theorem 1 it can be assumed that $k \leq 2^{\mathtt{vc}}$, as multiple centers that have the same partial assignment are redundant. However, this would increase the running time as a function of the parameter to doubly exponential, therefore we state the upper bounds with explicit dependence on k. It is, on the other hand, an interesting open question, whether d in the parameter is necessary for the algorithm parameterized by $\mathtt{tw}$. As shown by [20], this is not necessary for k-Means with Missing Entries, since the problem admits an FPT algorithm when parameterized by treewidth alone. Yet, it does not seem that the dynamic programming approaches used in their work and in our work, can be improved to avoid the factor of $d^{\mathcal{O}(\mathtt{tw})}$ in the case of k-Center with Missing Entries. Therefore, it is natural to ask whether it can be shown that k-Center with Missing Entries is W[1]-hard in this parameterization.

Another intriguing open question is the tightness of our algorithm in the parameterization by the vertex cover number (and fracture number). On the one hand, the running time we show is $2^{\mathtt{vc}^{2+o(1)}} \cdot \mathrm{poly}(nm)$ (for values of k in $\mathcal{O}(vc)$), exceeding the "natural" single-exponential in $\mathtt{vc}$ time, and we are not aware of a matching lower bound that is based on standard complexity assumptions. On the other hand, we show that improving this running time improves also the best-known running time for ILP Feasibility when parameterized by the number of constraints and Closest String parameterized by the number of strings. The latter are major open questions; [48] show also that several other open problems are equivalent to these. On the positive side, our algorithm in fact reduces k-Center with Missing Entries to just $2^{\mathtt{vc}}$ instances of ILP Feasibility. Since practical ILP solvers are quite efficient, coupling our reduction with an ILP solver is likely to result in the running time that is much more efficient than prescribed by the upper bound of Theorem 1.

References

1 Amir Abboud, Nick Fischer, Elazar Goldenberg, Karthik C. S., and Ron Safier. Can you solve closest string faster than exhaustive search? In *31st Annual European Symposium on Algorithms (ESA 2023)*, volume 274 of *LIPIcs*, pages 3:1–3:17, 2023. doi:10.4230/LIPIcs.ESA.2023.3.

2 Daniel Aloise, Amit Deshpande, Pierre Hansen, and Preyas Popat. Np-hardness of euclidean sum-of-squares clustering. *Mach. Learn.*, 75(2):245–248, 2009. doi:10.1007/S10994-009-5103-0.

3 Daniel N. Baker, Vladimir Braverman, Lingxiao Huang, Shaofeng H.-C. Jiang, Robert Krauthgamer, and Xuan Wu. Coresets for clustering in graphs of bounded treewidth. In *Proceedings of the 37th International Conference on Machine Learning (ICML 2020)*, volume 119 of *Proceedings of Machine Learning Research*, pages 569–579, 2020. URL: http://proceedings.mlr.press/v119/baker20a.html.

4 Sayan Bandyapadhyay, Fedor V. Fomin, and Kirill Simonov. On coresets for fair clustering in metric and euclidean spaces and their applications. *J. Comput. Syst. Sci.*, 142:103506, 2024. doi:10.1016/J.JCSS.2024.103506.

5 Hans L. Bodlaender, Tesshu Hanaka, Yasuaki Kobayashi, Yusuke Kobayashi, Yoshio Okamoto, Yota Otachi, and Tom C. van der Zanden. Subgraph isomorphism on graph classes that exclude a substructure. *Algorithmica*, 82:3566–3587, 2020. doi:10.1007/S00453-020-00737-Z.

6 Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved parameterized upper bounds for vertex cover. In *Mathematical Foundations of Computer Science 2006*, pages 238–249. Springer Berlin Heidelberg, 2006. doi:10.1007/11821069_21.

7 Raymond Chen. On mentzer's hardness of the k-center problem on the euclidean plane. Technical Report 383, Dartmouth College, 2021. Technical Report. URL: https://digitalcommons.dartmouth.edu/cs_tr/383/.

8 Zhi-Zhong Chen, Bin Ma, and Lusheng Wang. Randomized and parameterized algorithms for the closest string problem. In *Combinatorial Pattern Matching - 25th Annual Symposium, CPM 2014, Moscow, Russia, June 16-18, 2014. Proceedings*, volume 8486, pages 100–109. Springer, 2014. doi:10.1007/978-3-319-07566-2_11.

9 Vincent Cohen-Addad, Kasper Green Larsen, David Saulpic, and Chris Schwiegelshohn. Towards optimal lower bounds for k-median and k-means coresets. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2022)*, pages 1038–1051, Rome, Italy, 2022. doi:10.1145/3519935.3519946.

10 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

11 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.

12 Pavel Dvorák, Eduard Eiben, Robert Ganian, Dusan Knop, and Sebastian Ordyniak. The complexity landscape of decompositional parameters for ILP: programs with few global variables and constraints. *Artif. Intell.*, 300:103561, 2021. doi:10.1016/J.ARTINT.2021.103561.

13 Eduard Eiben, Fedor V. Fomin, Petr A. Golovach, William Lochet, Fahad Panolan, and Kirill Simonov. EPTAS for k-means clustering of affine subspaces. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2649–2659, 2021. doi:10.1137/1.9781611976465.157.

14 Eduard Eiben, Robert Ganian, Iyad Kanj, Sebastian Ordyniak, and Stefan Szeider. From Data Completion to Problems on Hypercubes: A Parameterized Analysis of the Independent Set Problem. In *18th International Symposium on Parameterized and Exact Computation (IPEC 2023)*, volume 285, pages 16:1–16:14, 2023. doi:10.4230/LIPIcs.IPEC.2023.16.

15 Eduard Eiben, Robert Ganian, Iyad Kanj, Sebastian Ordyniak, and Stefan Szeider. On the parameterized complexity of clustering problems for incomplete data. *J. Comput. Syst. Sci.*, 134:1–19, 2023. doi:10.1016/J.JCSS.2022.12.001.

16 Friedrich Eisenbrand and Robert Weismantel. Proximity results and faster algorithms for integer programming using the steinitz lemma. *ACM Trans. Algorithms*, 16:5:1–5:14, 2020. doi:10.1145/3340322.

17 Tomás Feder and Daniel H. Greene. Optimal algorithms for approximate clustering. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing (STOC)*, pages 434–444, 1988. doi:10.1145/62212.62255.

18 Uriel Feige. Np-hardness of hypercube 2-segmentation. *CoRR*, abs/1411.0821, 2014. arXiv:1411.0821.

19 Moti Frances and Ami Litman. On covering problems of codes. *Theory Comput. Syst.*, 30(2):113–119, 1997. doi:10.1007/S002240000044.

20 Robert Ganian, Thekla Hamm, Viktoriia Korchemna, Karolina Okrasa, and Kirill Simonov. The complexity of k-means clustering when little is known. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162, pages 6960–6987, 2022. URL: https://proceedings.mlr.press/v162/ganian22a.html.

21 Robert Ganian, Iyad A. Kanj, Sebastian Ordyniak, and Stefan Szeider. Parameterized algorithms for the matrix completion problem. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80, pages 1642–1651, 2018. URL: http://proceedings.mlr.press/v80/ganian18a.html.

22 Robert Ganian, Sebastian Ordyniak, and M. S. Ramanujan. On structural parameterizations of the edge disjoint paths problem. *Algorithmica*, 83:1605–1637, 2021. doi:10.1007/S00453-020-00795-3.

23 Leszek Gasieniec, Jesper Jansson, and Andrzej Lingas. Efficient approximation algorithms for the hamming center problem. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1999)*, pages 905–906. ACM/SIAM, 1999. URL: http://dl.acm.org/citation.cfm?id=314500.315081.

24 Tomáš Gavenčiak, Martin Koutecký, and Dušan Knop. Integer programming in parameterized complexity: Five miniatures. *Discrete Optimization*, 44:100596, 2022. doi:10.1016/j.disopt.2020.100596.

25 Rong Ge, Martin Ester, Byron J. Gao, Zengjian Hu, Binay Bhattacharya, and Boaz Ben-Moshe. Joint cluster analysis of attribute data and relationship data: The connected k-center problem, algorithms and applications. *ACM Transactions on Knowledge Discovery from Data*, 2(2):1–35, 2008. doi:10.1145/1376815.1376816.

26 Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985. doi:10.1016/0304-3975(85)90224-5.

27 Jens Gramm, Rolf Niedermeier, and Peter Rossmanith. Fixed-parameter algorithms for CLOSEST STRING and related problems. *Algorithmica*, pages 25–42, 2003. doi:10.1007/S00453-003-1028-3.

28 Pierre Hansen and Brigitte Jaumard. Cluster analysis and mathematical programming. *Mathematical Programming*, 79(1):191–215, 1997. doi:10.1007/BF02614317.

29 Danny Hermelin and Liat Rozenberg. Parameterized complexity analysis for the closest string with wildcards problem. *Theoretical Computer Science*, pages 11–18, 2015. doi:10.1016/j.tcs.2015.06.043.

30 Wen-Lian Hsu and George L. Nemhauser. Easy and hard bottleneck location problems. *Discrete Applied Mathematics*, 1(3):209–215, 1979. doi:10.1016/0166-218X(79)90044-1.

31 Debajyoti Kar, Mert Kosan, Debmalya Mandal, Sourav Medya, Arlei Silva, Palash Dey, and Swagato Sanyal. Feature-based individual fairness in k-clustering. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023)*, pages 2772–2774, 2023. doi:10.5555/3545946.3599073.

32 Ioannis Katsikarelis, Michael Lampis, and Vangelis Th. Paschos. Structural parameters, tight bounds, and approximation for (k,r)-center. *Discrete Applied Mathematics*, pages 90–117, 2019. doi:10.1016/j.dam.2018.11.002.

33 Dusan Knop, Martin Koutecký, and Matthias Mnich. Combinatorial n-fold integer programming and applications. *Math. Program.*, 184:1–34, 2020. doi:10.1007/S10107-019-01402-2.

34 Dusan Knop, Michal Pilipczuk, and Marcin Wrochna. Tight complexity lower bounds for integer linear programming with few constraints. *ACM Trans. Comput. Theory*, 12:19:1–19:19, 2020. doi:10.1145/3397484.

35 Yuval Kochman, Arya Mazumdar, and Yury Polyanskiy. The adversarial joint source-channel problem. In *Proceedings of the 2012 IEEE International Symposium on Information Theory, ISIT 2012, Cambridge, MA, USA, July 1-6, 2012*, pages 2112–2116. IEEE, 2012. doi:10.1109/ISIT.2012.6283735.

36 Michael Lampis and Valia Mitsou. Fine-grained meta-theorems for vertex integrity. *Log. Methods Comput. Sci.*, 20, 2024. doi:10.46298/LMCS-20(4:18)2024.

37 J. Kevin Lanctôt, Ming Li, Bin Ma, Shaojiu Wang, and Louxin Zhang. Distinguishing string selection problems. *Inf. Comput.*, 185(1):41–55, 2003. doi:10.1016/S0890-5401(03)00057-9.

38 Ming Li, Bin Ma, and Lusheng Wang. On the closest string and substring problems. *J. ACM*, 49(2):157–171, 2002. doi:10.1145/506147.506150.

39 Stuart P. Lloyd. Least squares quantization in pcm. *IEEE Trans. Inf. Theory*, 28(2):129–136, 1982. doi:10.1109/TIT.1982.1056489.

40 Bin Ma and Xiaoming Sun. More efficient algorithms for closest string and substring problems. In *Research in Computational Molecular Biology, 12th Annual International Conference, RECOMB 2008, Singapore, March 30 - April 2, 2008. Proceedings*, volume 4955, pages 396–409. Springer, 2008. doi:10.1007/978-3-540-78839-3_33.

41 Bin Ma and Xiaoming Sun. More efficient algorithms for closest string and substring problems. *SIAM J. Comput.*, pages 1432–1443, 2009. doi:10.1137/080739069.

42 Meena Mahajan, Prajakta Nimbhorkar, and Kasturi Varadarajan. The planar k-means problem is np-hard. In *Proceedings of the 3rd International Workshop on Algorithms and Computation (WALCOM)*, pages 274–285, 2009. doi:10.1007/978-3-642-00202-1_24.

43 Arya Mazumdar, Yury Polyanskiy, and Barna Saha. On chebyshev radius of a set in hamming space and the closest string problem. In *Proceedings of the 2013 IEEE International Symposium on Information Theory, Istanbul, Turkey*, pages 1401–1405. IEEE, 2013. doi:10.1109/ISIT.2013.6620457.

44 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.

45 Stuart G. Mentzer. Approximability of metric clustering problems, 2016. Unpublished manuscript, March.

46 Jacob Munson, Breschine Cummins, and Dominique Zosso. An introduction to collaborative filtering through the lens of the netflix prize. *Knowledge and Information Systems*, 67:3049–3098, 2025. doi:10.1007/s10115-024-02315-z.

47 Murray Patterson, Tobias Marschall, Nadia Pisanti, Leo van Iersel, Leen Stougie, Gunnar W. Klau, and Alexander Schönhuth. WhatsHap: Weighted haplotype assembly for future-generation sequencing reads. *Journal of Computational Biology*, 22(6):498–509, February 2015. doi:10.1089/CMB.2014.0157.

48 Lars Rohwedder and Karol Węgrzycki. Fine-Grained Equivalence for Problems Related to Integer Linear Programming. In *16th Innovations in Theoretical Computer Science Conference (ITCS 2025)*, volume 325, pages 83:1–83:18, 2025. doi:10.4230/LIPIcs.ITCS.2025.83.

49 Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000. doi:10.1109/34.868688.

50 Nikola Stojanovic, Piotr Berman, Deborah Gumucio, Ross Hardison, and Webb Miller. A linear-time algorithm for the 1-mismatch problem. In *Proceedings of the 5th International Workshop on Algorithms and Data Structures*, WADS '97, pages 126–135. Springer-Verlag, 1997. doi:10.1007/3-540-63307-3_53.

51 Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. Data mining cluster analysis: basic concepts and algorithms. *Introduction to Data Mining*, 487:533, 2013.

52 Xiaoliang Wu, Qilong Feng, Ziyun Huang, Jinhui Xu, and Jianxin Wang. New algorithms for distributed fair k-center clustering: Almost accurate as sequential algorithms. In *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024)*, pages 1938–1946, 2024. doi:10.5555/3635637.3663057.

An ETH-Tight FPT Algorithm for Rejection-Proof Set Packing with Applications to Kidney Exchange

Bart M. P. Jansen
Eindhoven University of Technology, The Netherlands

Jeroen S. K. Lamme
Eindhoven University of Technology, The Netherlands

Ruben F. A. Verhaegh
Eindhoven University of Technology, The Netherlands

Abstract

We study the parameterized complexity of a recently introduced multi-agent variant of the Kidney Exchange problem. Given a directed graph G and integers d and k, the standard problem asks whether G contains a packing of vertex-disjoint cycles, each of length $\leq d$, covering at least k vertices in total. In the multi-agent setting we consider, the vertex set is partitioned over several agents who reject a cycle packing as solution if it can be modified into an alternative packing that covers more of their own vertices. A cycle packing is called *rejection-proof* if no agent rejects it and the problem asks whether such a packing exists that covers at least k vertices.

We exploit the sunflower lemma on a set packing formulation of the problem to give a kernel for this Σ_2^P-complete problem that is polynomial in k for all constant values of d. We also provide a $2^{\mathcal{O}(k \log k)} + n^{\mathcal{O}(1)}$ algorithm based on it and show that this FPT algorithm is asymptotically optimal under the ETH. Further, we generalize the problem by including an additional positive integer c in the input that naturally captures how much agents can modify a given cycle packing to reject it. For every constant c, the resulting problem simplifies from being Σ_2^P-complete to NP-complete. The super-exponential lower bound already holds for $c = 2$, though. We present an ad-hoc single-exponential algorithm for $c = 1$. These results reveal an interesting discrepancy between the classical and parameterized complexity of the problem and give a good view of what makes it hard.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Applied computing → Health informatics; Theory of computation → Algorithmic game theory; Mathematics of computing → Combinatorial algorithms

Keywords and phrases Parameterized complexity, Multi-agent kidney exchange, Kernelization, Set packing

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.9

Related Version *Full Version*: `https://arxiv.org/abs/2509.11965`

Funding *Bart M. P. Jansen*: Supported by the Dutch Research Council (NWO) through Gravitation-grant NETWORKS-024.002.003.
Jeroen S. K. Lamme: Supported by the Dutch Research Council (NWO) through Gravitation-grant NETWORKS-024.002.003.

1 Introduction

The goal of this paper is to analyze the parameterized complexity of algorithmic problems that arise from a recently introduced multi-party variant of kidney exchange problems. Before describing our results, we first present some relevant context.

For patients with severely reduced kidney function, transplantation is often the preferred treatment [7]. As it is possible to live with only one healthy kidney, live kidney donation is employed to reduce waiting times for sparsely available donor organs. In this setting, a

© Bart M. P. Jansen, Jeroen S. K. Lamme, and Ruben F. A. Verhaegh;
licensed under Creative Commons License CC-BY 4.0

willing donor offers one of their kidneys for transplantation to a patient. The patient and donor must be medically compatible for a successful transplantation. When the patient and donor are not compatible, they can opt to participate in a kidney exchange program (KEP).

Such programs aim to link multiple patient-donor pairs so that a donor from one pair is compatible with a patient from another pair. Their goal is to group sets of patient-donor pairs, such that within each group, a cyclic order of compatibility exists. Performing this cycle of transplants means that donors only undergo surgery if their corresponding patient receives a kidney in that exchange. Individual programs each have their own additional restrictions [5]. It is common to restrict the length of transplantation cycles to some constant d, due to practical considerations which include doing the associated surgeries simultaneously.

We study these KEPs from an algorithmic perspective. Given a set of participating patient-donor pairs and the medical compatibilities between them, the algorithmic task becomes to find a combination of transplantation cycles that maximizes the total number of patients receiving a healthy kidney. We can model each patient-donor pair as a vertex and the compatibility of the donor of a pair a to the patient of a pair b as a directed edge from a to b. We call a cycle of length at most d a *d-cycle* and use this notation to obtain the following algorithmic graph problem that describes KEPs in one of their most basic forms.

Kidney Exchange with d-cycles (KE-d)	
Input:	A directed graph G and an integer k.
Task:	Determine whether there is a set $\mathcal{C}$ of pairwise vertex-disjoint d-cycles such that $\mathcal{C}$ covers at least k vertices.

This problem has already been studied from the perspective of parameterized algorithms in the literature, with fixed-parameter tractable (FPT) algorithms w.r.t. the input parameter k and the number of neighborhood equivalence classes as examples [16, 20].

Multi-agent KEPs. Extensions of the standard kidney exchange problem have also been studied. In this work, we consider one such extension which we now motivate and introduce. It is known that the fraction of patients to receive a healthy kidney can increase significantly with an increase in the total number of participating pairs [2]. This motivates collaborations between multiple programs, from now on referred to as agents. Such collaborations, however, can be hindered by a mismatch between the individual interests of agents, as these need not align with the collective goal of helping as many patients as possible overall. In particular, a socially optimal solution – in which as many patients as possible are treated overall – is not guaranteed to help as many people from each individual agent as that agent could have achieved on their own. This may stop agents from participating in collaborative programs or, when they do participate, move them to withhold information from the collective.

Several ways of incorporating fairness into multi-agent KEPs have been studied [2, 3, 4, 8, 9, 15, 19]. Recently, Blom, Smeulders, and Spieksma [6] also introduced a framework to capture fairness via a game-theoretical concept of *rejections*. They translate this into an algorithmic task that asks for a solution that is not rejected by any agent. We study the resulting algorithmic task. To introduce it, we first define the concept of rejections.

Recall the original KE-d problem on directed graphs. To encode multiple agents into the input, we partition the vertex set of the input graph over the agents. In the obtained graph, we refer to a cycle that only contains vertices from a single agent i as an *i-internal* cycle. Like in the original KE-d problem, a solution consists of a packing $\mathcal{C}$ of pairwise vertex-disjoint d-cycles. A given such packing $\mathcal{C}$ will be rejected by an agent under the conditions listed in the definition below. See also Figure 1 for an example, adapted from a figure by Blom et al.

Figure 1 A socially optimal packing of cycles is shown on the left and covers six vertices. However, the blue agent rejects this solution as they could modify it to cover more blue vertices as shown on the right. They can do this by rejecting the left-most 3-cycle and including an internal 2-cycle.

▶ **Definition 1.1.** *Let G be a graph, where $V(G) = V_1 \cup \ldots \cup V_p$ is partitioned over p different agents. Let d be an upper bound on the length of allowed transplantation cycles. Let $\mathcal{C}$ be a set of pairwise vertex-disjoint d-cycles in G. If there is an agent $i \in [p]$ for which there exist*

- *a subset $\mathcal{C}_{\text{rej}} \subseteq \mathcal{C}$, and*
- *a set $\mathcal{C}_{\text{int}}$ of i-internal d-cycles, such that $\mathcal{C}' := (\mathcal{C} \setminus \mathcal{C}_{\text{rej}}) \cup \mathcal{C}_{\text{int}}$ is a set of cycles that are still pairwise vertex-disjoint while $\mathcal{C}'$ covers more vertices from V_i than $\mathcal{C}$ does,*

then, agent i is said to reject *the packing $\mathcal{C}$. Equivalently, we may say that agent i specifically rejects the cycles in $\mathcal{C}_{\text{rej}}$ and we refer to $\mathcal{C}'$ as an* alternative packing *by which agent i rejects.*

This definition captures that the rejecting agent i could modify a given cycle packing $\mathcal{C}$ into another packing $\mathcal{C}'$ that benefits more of their own patients by only making changes pertaining to patients within their control. We call a packing that is not rejected by any agent *rejection-proof*. Then, a guarantee to always find such a solution also serves as a guarantee for agents that participation in the joint program is never detrimental to their personal interests. This led Blom et al. [6] to introduce the following problem, whose parameterized complexity we investigate in this work.

Rejection-Proof Kidney Exchange with d-cycles (RPKE-d)	
Input:	A directed graph G where $V(G) = V_1 \cup \ldots \cup V_p$ is partitioned among p agents, and an integer k.
Task:	Determine whether there is a *rejection-proof* set $\mathcal{C}$ of pairwise vertex-disjoint d-cycles such that $\mathcal{C}$ covers at least k vertices.

For ease of notation, we refer to any packing of pairwise vertex-disjoint d-cycles that covers at least k vertices in an RPKE-d instance as a *candidate solution*. Hence, a packing of d-cycles forms a solution to an instance if it is both a candidate solution and rejection-proof.

The above problem takes a leap in complexity compared to the single-agent problem KE-d. Whereas KE-d is known to be NP-complete for all finite $d \geq 3$ [1], Blom et al. have shown RPKE-d to be Σ_2^P-complete for $d \geq 3$, even when there are only two participating agents [6, Theorem 1]. In fact, it can be shown that checking whether a given agent rejects a given candidate solution is already NP-hard since this is at least as hard as checking whether a given packing is optimal in the NP-hard setting of only one agent. Partly motivated by this observation, we also consider a more restricted definition of rejections.

For some integer $c \geq 0$, we say that an agent *c-rejects* a packing $\mathcal{C}$ if the rejection criterion from Definition 1.1 can be satisfied with $|\mathcal{C}_{\text{rej}}| \leq c$. We call a packing that no agent c-rejects *c-rejection-proof*. Now, for each pair of positive integers c and d we define the problem c-Rejection-Proof Kidney Exchange with d-cycles (c-RPKE-d) as the variant of the above RPKE-d problem in which we aim to determine whether the input contains a candidate solution that is c-rejection-proof.

The integer c in this definition indicates some level of complexity that a rejection can have. Additionally, it provides a way to interpolate between the problems KE-d ($c = 0$) and RPKE-d ($c = \infty$). More importantly, for any constant c, we observe that the task of

determining whether a given agent c-rejects a given packing $\mathcal{C}$ of d-cycles is polynomial-time solvable when d is also constant. This is witnessed by the fact that, if an agent c-rejects the packing $\mathcal{C}$, an alternative packing can always be obtained by rejecting a constant number of cycles and including a constant number of internal cycles. It follows that c-RPKE-d is contained in NP when c and d are constants, as opposed to the Σ_2^P-complete RPKE-d problem that does not limit the number of cycles that agents are allowed to reject.

Our contribution. We study the complexity of RPKE-d and c-RPKE-d parameterized by k, the number of vertices to be covered, and we present both positive and negative results. We prove our positive results on related set packing problems that generalize the two graph problems. This allows us to use some of the tools developed for such problems. Prior to our work, the only positive result known for RPKE-d was a polynomial-time algorithm for $d = 2$ [8].

We give an algorithm that, for constant values of d, solves RPKE-d in $2^{\mathcal{O}(k \log k)} + n^{\mathcal{O}(1)}$ time on n-vertex graphs. It consists of two parts, the first of which is a kernelization algorithm. It takes an arbitrary instance as input and outputs an equivalent instance of size $\mathcal{O}\left((k^{d+1})^d\right)$ in polynomial time, via a careful agent-aware application of the sunflower lemma [12] to eliminate transplantation cycles. In set packing problems, removing sets during preprocessing typically comes at the risk of removing sets that are required to make an optimal solution, thereby possibly transforming YES-instances into NO-instances. The sunflower lemma provides a tool to circumvent this. Our rejection-proof problem settings introduce an additional type of behavior that needs care: transplantation cycles that do not belong to optimal solutions may still play a role in rejections. Removing the wrong cycles may cause an agent not to reject a certain candidate solution that it otherwise would have. We present a two-step reduction approach guaranteed to preserve the answer. The second part of the algorithm consists of brute-force on the reduced instance.

We give a reduction from Subgraph Isomorphism to show that, under the Exponential Time Hypothesis (ETH), the algorithm is asymptotically optimal in terms of n and k. We even show the stronger bound (independent of k) that a $2^{o(n \log n)}$ time algorithm for RPKE-d or c-RPKE-d would violate the ETH for any $c \geq 2$ and $d \geq 3$. We exploit the mechanism of rejections to facilitate our reduction from the Subgraph Isomorphism problem, for which such a superexponential ETH-based lower bound is known [10]. This gives the rejection-proof kidney exchange problems a similar complexity status to other problems in the literature, such as Metric Dimension and Geodetic Set parameterized by vertex cover number, for which a brute-force algorithm on a polynomial kernel is also ETH-optimal [14].

To complement the ETH-hardness of c-RPKE-d with $c \geq 2$, we give a $3^n \cdot n^{\mathcal{O}(1)}$ time algorithm for the problem when $c = 1$. This allows us to see surprising jumps in the algorithmic complexity of c-RPKE-d for varying values of c. While no big difference in complexity is obtained between $c = 1$ and $c = 0$ (in which case the problem can be solved in $2^n \cdot n^{\mathcal{O}(1)}$ time with a standard application of dynamic programming over subsets [10, Chapter 6]), our results show that the problem does become computationally harder from a parameterized perspective when $c = 2$ (assuming the ETH). We also see that no more jumps in parameterized complexity appear for larger values of c. The surprising aspect here is that this one discrete jump in the parameterized complexity of c-RPKE-d does not coincide with the jump in the classical complexity of the problem between being contained in NP vs. not being contained in NP: we have seen that, for constant $d \geq 3$, the c-RPKE-d problem is contained in NP for *every* constant value of c, while the RPKE-d problem is Σ_2^P-complete.

Another jump in complexity is observed when looking at the number of vertices that are *not* covered by a solution. When asked to find a solution that covers all vertices in the input, the problem again collapses to the NP-complete packing problem KE-d. However, our ETH-hardness proof shows that finding a solution that covers all but one vertex is essentially as difficult as the general problem in terms of running time, assuming the ETH. Combined, our results yield a good understanding of what makes rejection-proof kidney exchange hard.

Organization. The remainder of this work is organized as follows. We start with some preliminaries in Section 2. Next, we give a polynomial kernel and an FPT algorithm for a set packing generalization of RPKE-d in Section 3, and we present a matching ETH-lower bound in Section 4. In Section 5, we give a single-exponential algorithm under 1-rejections and we conclude in Section 6. The proofs of statements marked (★) are deferred to the full version of this paper.

2 Preliminaries

All graphs considered are directed, unless stated otherwise. We use standard notation for graphs and concepts from parameterized complexity; see the textbook by Cygan et al. [11]. We write $\langle v_1, \ldots, v_\ell \rangle$ to denote the directed cycle with edge set $\{(v_1, v_2), \ldots, (v_{\ell-1}, v_\ell), (v_\ell, v_1)\}$. We use the term *packing* to refer to a collection of sets that satisfies certain pairwise disjointness conditions, such as collections of vertex-disjoint cycles in a graph or collections of pairwise-disjoint subsets of a universe. Since the disjointness conditions differ depending on the context, we always specify which condition holds for the packing at hand.

A set system is a pair $(U, \mathcal{S})$ where U is a universe of elements and $\mathcal{S} \subseteq 2^U$ is a collection of subsets of this universe. For a subset $U' \subseteq U$, we use the notation $\mathcal{S}[U']$ to denote the collection of sets from $\mathcal{S}$ that contain only elements of U'. A *hitting set* of $\mathcal{S}$ is a subset of U that intersects each set $S \in \mathcal{S}$. We use the following folklore lemma.

▶ **Lemma 2.1 (★).** *For a set system $(U, \mathcal{S})$, it can be checked in $2^{|U|} \cdot (|U| + |\mathcal{S}|)^{\mathcal{O}(1)}$ time whether $\mathcal{S}$ contains a collection of pairwise disjoint sets that covers all elements of U.*

Next, we present the sunflower lemma by Erdős and Rado [12] and a related definition of *sunflowers* in set systems. In a set system $(U, \mathcal{S})$, a sunflower $\mathcal{F}$ with z petals and *core* Y is a collection of sets $S_1, \ldots, S_z \in \mathcal{S}$ such that $S_i \cap S_j = Y$ for all $i \neq j$. We refer to a set $S_i \setminus Y$ as the *petal* corresponding to set S_i and we require all petals to be non-empty. We state a variant of it as proven in the textbook by Flum and Grohe [13, page 212].

▶ **Lemma 2.2** (Sunflower lemma). *Let $(U, \mathcal{S})$ be a set system, such that each set in $\mathcal{S}$ has size at most d. If $|\mathcal{S}| > d \cdot d!(z-1)^d$, then the set system contains a sunflower with z petals and such a sunflower can be computed in time polynomial in $|U|$, $|\mathcal{S}|$, and z.*

The following lemma follows from the definitions of sunflowers and hitting sets.

▶ **Lemma 2.3 (★).** *Let $(U, \mathcal{S})$ be a set system with sets of size $\leq d$ for which there exists a hitting set of size $\leq h$, for some integers d, h. Let $\mathcal{X} \subseteq \mathcal{S}$ be a collection of disjoint sets and let $\mathcal{F} \subseteq \mathcal{S}$ be a sunflower of size $d(h-1)+2$ that contains one of the sets $X \in \mathcal{X}$. Then, there is a set $X' \in \mathcal{F} \setminus \{X\}$ such that $(\mathcal{X} \setminus X) \cup \{X'\}$ is a collection of pairwise disjoint sets.*

When analyzing the running time of an algorithm whose input is a graph, we denote the number of vertices and edges in the input graph by n and m, respectively. For algorithms operating on a set system, we use n to denote the number of elements in the universe and m for the total number of sets in the system.

Rejection-proof set packing. As mentioned, we present our positive results for problems that generalize the kidney exchange problems defined in the introduction. Intuitively, one may think of translating an RPKE-d instance on a graph G into this generalized setting by constructing a set system $(U, \mathcal{S})$ with $U = V(G)$ and including a set in $\mathcal{S}$ for every vertex set that forms a d-cycle in G. Related definitions follow analogously, but we briefly reiterate and name them below. To represent p different agents in a set system $(U, \mathcal{S})$, the universe $U = U_1 \cup \ldots \cup U_p$ is partitioned into p sets. A set $S \in \mathcal{S}$ is called *i-internal* if $S \subseteq U_i$.

▶ **Definition 2.4.** *Let $(U, \mathcal{S})$ be a set system such that the universe $U = U_1 \cup \ldots \cup U_p$ is partitioned over p different agents. Let $\mathcal{X} \subseteq \mathcal{S}$ be a packing of pairwise disjoint sets. Suppose there is an agent $i \in [p]$ for which there exist:*

- *a subset $\mathcal{X}_{\text{rej}} \subseteq \mathcal{X}$, and*
- *a set $\mathcal{X}_{\text{int}}$ of i-internal sets such that $\mathcal{X}' := (\mathcal{X} \setminus \mathcal{X}_{\text{rej}}) \cup \mathcal{X}_{\text{int}}$ is a collection of sets that are still pairwise disjoint, while $\mathcal{X}'$ covers more elements from U_i than $\mathcal{X}$ does,*

then, agent i is said to reject *the packing $\mathcal{X}$. Equivalently, we may say that agent i specifically rejects the sets in $\mathcal{X}_{\text{rej}}$ and we refer to $\mathcal{X}'$ as an alternative packing by which agent i rejects.*

We call a packing that is not rejected by any agent *rejection-proof*, and we use this notation to introduce the following generalized problem statement.

Rejection-Proof d-Set Packing (RP-d-SP)	
Input:	A universe $U = U_1 \cup \ldots \cup U_p$ that is partitioned over p agents, a collection $\mathcal{S} \subseteq 2^U$ of sets of size $\leq d$, and an integer k.
Task:	Determine whether there is a rejection-proof collection $\mathcal{X} \subseteq \mathcal{S}$ of disjoint sets that covers at least k elements from U.

Like in the graph setting, we call any packing of pairwise disjoint sets that cover at least k vertices a *candidate solution*. Further, for some integer $c \geq 0$, we say that an agent *c-rejects* a given packing $\mathcal{X}$ if the rejection criterion from Definition 2.4 can be satisfied with $|\mathcal{X}_{\text{rej}}| \leq c$. A packing that no agent c-rejects is called *c-rejection-proof*. We define c-Rejection-Proof d-Set Packing (c-RP-d-SP) as a problem with the same input as RP-d-SP, but in which the goal is to determine whether the input contains a c-rejection-proof candidate solution.

Exponential Time Hypothesis (ETH). In Section 4, we derive a hardness result that is conditional on the well-known ETH. This is a technical conjecture implying that the satisfiability of Boolean 3-CNF formulas cannot be checked in subexponential time [17]. Specifically, we use a known hardness result for the Subgraph Isomorphism problem conditional on the ETH. This problem takes two undirected graphs G and H as input and asks whether H is isomorphic to a subgraph of G. More formally, the problem asks whether there is an injective function $\varphi : V(H) \to V(G)$ such that $\{u, v\} \in E(H)$ implies $\{\varphi(u), \varphi(v)\} \in E(G)$. Such a function is called a subgraph isomorphism from H to G. The following conditional lower bound on the computational complexity of this problem is known.

▶ **Lemma 2.5** ([10]). *Assuming the ETH, no algorithm exists that determines in $2^{o(n_G \log n_G)}$ time whether an undirected graph H is isomorphic to a subgraph of G, where $n_G = |V(G)|$.*

3 A kernelization and FPT algorithm for rejection-proof packings

In this section, we present a polynomial kernel and a derived FPT algorithm for RP-d-SP, which translates directly to RPKE-d as explained in Section 2. We start with two simple statements that are useful in proving the correctness of our kernel.

▶ **Lemma 3.1.** *Let $I_1 = (U_1, \ldots, U_p, \mathcal{S}, k)$ be an RP-d-SP instance, let $\mathcal{X} \subseteq \mathcal{S}$ be a solution for it and let $X \in \mathcal{S}$. If $X \notin \mathcal{X}$, then $\mathcal{S}$ is a solution for the instance $I_2 = (U_1, \ldots, U_p, \mathcal{S} \setminus \{X\}, k)$.*

Proof. Clearly, $\mathcal{X}$ is also a candidate solution in I_2. Now, suppose for contradiction that some agent i rejects $\mathcal{X}$ by proposing an alternative packing $(\mathcal{X} \setminus \mathcal{X}_{\text{rej}}) \cup \mathcal{X}_{\text{int}}$. This alternative packing can also be constructed in I_1, contradicting that $\mathcal{X}$ is rejection-proof in I_1. ◀

▶ **Observation 3.2.** *Let $S_i = \bigcup_{S \in \mathcal{S}[U_i]} S$ denote the elements which are contained in i-internal sets for some agent $i \in [p]$. If for collections $\mathcal{X}$ and $\mathcal{X}'$ it holds that $\{X \in \mathcal{X} \mid X \cap S_i \neq \emptyset\} = \{X' \in \mathcal{X}' \mid X' \cap S_i \neq \emptyset\}$, then i rejects $\mathcal{X}$ if and only if i rejects $\mathcal{X}'$.*

The above observation captures that an agent i cannot gain anything by rejecting a set that is not intersected by an i-internal set. Thus, we can assume they will not reject such sets. $\mathcal{X}$ and $\mathcal{X}'$ become identical after removing these sets, so the observation is correct.

Now, we present two reduction rules that act on an input instance $(U_1, \ldots, U_p, \mathcal{S}, k)$ of RP-d-SP and prove that they are safe. A reduction rule is *safe* if its application preserves the YES/NO answer to the decision problem. The first rule aims to reduce the number of internal sets.

▶ **Reduction rule 1.** *Let $\mathcal{F} \subseteq \mathcal{S}[U_i]$, for some $i \in [p]$, be a sunflower of size $d(k \cdot d - 1) + 2$. Remove a set $F \in \mathcal{F}$ of minimum size.*

▶ **Lemma 3.3.** *Reduction rule 1 is safe if $\mathcal{S}$ has a hitting set of size $\leq k \cdot d$.*

Proof. Let $I = (U_1, \ldots, U_p, \mathcal{S}, k)$ be an instance of RP-d-SP where there exists some $i \in [p]$ and sunflower $\mathcal{F} \subseteq \mathcal{S}[U_i]$ of size $d(k \cdot d - 1) + 2$ with core Y. Let S be the set of minimum size in $\mathcal{F}$ that is removed by Reduction rule 1 to create the instance $I' = (U_1, \ldots, U_p, \mathcal{S} \setminus \{S\}, k)$. We will now argue that I is a YES-instance if and only if I' is a YES-instance.

(⇒). Assume I has a solution $\mathcal{X}$. We claim there is a solution for I that does not use S. If $S \notin \mathcal{X}$, our claim holds. Suppose $S \in \mathcal{X}$. Invoking Lemma 2.3 with $h = k \cdot d$ yields that a $S' \in \mathcal{F}$ exists such that the sets in $\mathcal{X}' = (\mathcal{X} \setminus \{S\}) \cup \{S'\}$ are pairwise disjoint. Since S is a minimum-size set in $\mathcal{F}$, it follows that $|S'| \geq |S|$, so the collection $\mathcal{X}'$ covers at least k elements. To show that $\mathcal{X}'$ is rejection-proof, suppose for contradiction that some agent j rejects $\mathcal{X}'$ by proposing the alternate packing $(\mathcal{X}' \setminus \mathcal{X}'_{\text{rej}}) \cup \mathcal{X}'_{\text{int}}$. We distinguish three cases:

- $j \neq i$. Since $S \in \mathcal{S}[U_i]$, each set $J \in \mathcal{S}$ with $J \cap U_j \neq \emptyset$ is contained in $\mathcal{X}'$ if and only if it is contained in $\mathcal{X}$. From Observation 3.2 it follows that agent j would reject $\mathcal{X}$.
- $j = i$ and $S' \in \mathcal{X}'_{\text{rej}}$. Let $\mathcal{X}_{\text{rej}} = (\mathcal{X}'_{\text{rej}} \setminus \{S'\}) \cup \{S\}$ and note that $\mathcal{X}' \setminus \mathcal{X}'_{\text{rej}} = \mathcal{X} \setminus \mathcal{X}_{\text{rej}}$. By the definition of rejection $(\mathcal{X} \setminus \mathcal{X}_{\text{rej}}) \cup \mathcal{X}'_{\text{int}}$ is a collection of disjoint sets. Both S and S' are j-internal and $|S| \leq |S'|$, hence $\mathcal{X}$ covers at most as many elements of U_j as $\mathcal{X}'$ does. By definition of rejection, $(\mathcal{X}' \setminus \mathcal{X}'_{\text{rej}}) \cup \mathcal{X}'_{\text{int}} = (\mathcal{X} \setminus \mathcal{X}_{\text{rej}}) \cup \mathcal{X}'_{\text{int}}$ covers more elements from U_j than $\mathcal{X}'$ does, thus also more than $\mathcal{X}$ covers. This would mean that $\mathcal{X}$ would be rejected by agent j by rejecting the sets $\mathcal{X}_{\text{rej}} \subseteq \mathcal{X}$ and replacing these with the sets $\mathcal{X}'_{\text{int}}$.
- $j = i$ and $S' \notin \mathcal{X}'_{\text{rej}}$. Let $\mathcal{X}_{\text{rej}} = \mathcal{X}'_{\text{rej}} \cup \{S\}$ and $\mathcal{X}_{\text{int}} = \mathcal{X}'_{\text{int}} \cup \{S'\}$. Observe that $\mathcal{X}_{\text{int}} \subseteq \mathcal{S}[U_j]$ and $(\mathcal{X} \setminus \mathcal{X}_{\text{rej}}) \cup \mathcal{X}_{\text{int}} = (\mathcal{X}' \setminus \mathcal{X}'_{\text{rej}}) \cup \mathcal{X}'_{\text{int}}$. Hence agent j would reject $\mathcal{X}$.

As all cases contradict the assumption that $\mathcal{X}$ is rejection-proof, $\mathcal{X}'$ is a solution for I that does not contain S. Now Lemma 3.1 implies that I' has a solution and is a YES-instance.

(⇐). Assume I' has a solution $\mathcal{X}'$. We claim that $\mathcal{X}'$ is also a solution for the instance I. It suffices to show that no rejection can take place where $S \in \mathcal{X}_{\text{int}}$, as the other requirements for $\mathcal{X}'$ to be a solution for I follow directly from $\mathcal{X}'$ being a solution for I'.

Assume, for sake of contradiction, that agent i rejects $\mathcal{X}'$ by rejecting $\mathcal{X}_{\text{rej}}$ and replacing these sets with $\mathcal{X}_{\text{int}}$, where $S \in \mathcal{X}_{\text{int}}$. Because $(\mathcal{X}' \setminus \mathcal{X}_{\text{rej}}) \cap \mathcal{X}_{\text{int}}$ is a collection of disjoint sets and $S \in \mathcal{F}$ it follows from Lemma 2.3 with $h = k \cdot d$ that there exists some set $S' \in \mathcal{F}$ such that $(((\mathcal{X}' \setminus \mathcal{X}_{\text{rej}}) \cap \mathcal{X}_{\text{int}}) \setminus \{S\}) \cup \{S'\}$ is a collection of disjoint sets. Note that $|S'| \geq |S|$ and $S, S' \in \mathcal{S}[U_i]$. Hence agent i can also reject $\mathcal{X}'$ as a solution for I by rejecting $\mathcal{X}_{\text{rej}}$ and replacing these sets with $(\mathcal{X}_{\text{int}} \setminus \{S\}) \cup \{S'\}$. Similarly, agent i can reject $\mathcal{X}'$ as a solution for I'; a contradiction. Hence $\mathcal{X}'$ is a solution for I, which means I is a YES-instance. ◀

Observe that in the forward direction of the proof we increase the number of sets that are rejected by some agent j. This is possible because an agent can reject an unbounded number of sets in RPKE-d. The proof as written does not work in the setting of c-RPKE-d. Our second reduction rule aims to reduce the number of sets that are not internal.

▶ **Reduction rule 2.** *Let $\mathcal{F} \subseteq \mathcal{S}$ be a sunflower with core Y of size $d(k \cdot d - 1) + 2$ such that:*

- *No petal $F \setminus Y$, for $F \in \mathcal{F}$, is intersected by an internal set.*
- *For each agent i that has an internal set that intersects Y, for all $F, F' \in \mathcal{F}$:*

$$|F \cap U_i| = |F' \cap U_i|.$$

Remove a set $F \in \mathcal{F}$ of minimum size.

▶ **Lemma 3.4 (★).** *Reduction rule 2 is safe if $\mathcal{S}$ has a hitting set of size $\leq k \cdot d$.*

The proof of the lemma follows the same structure as the proof of Lemma 3.3. The restrictions placed on the sunflower used in Reduction rule 2 together with Lemma 3.1 can be used to prove one direction of the proof. The other direction follows from the fact that any removed set is not internal and can thus not be used as a replacement set in a rejection.

To bound the kernel size we define $f_d(k) = d \cdot d!(d(k \cdot d - 1) + 1)^d$ and $g_d(k) = d \cdot d! \left(d^{d-1} \cdot (d(k \cdot d - 1) + 1) + d \cdot k \cdot f_d(k)\right)^d$. Note that $g_d(k) = \mathcal{O}\left(\left(k^{d+1}\right)^d\right)$ for fixed d.

▶ **Lemma 3.5 (★).** *If for an instance $I = (U_1, \ldots, U_p, \mathcal{S}, k)$ of RP-d-SP it holds that $|\mathcal{S}| \geq g_d(k)$, then one of the reduction rules can be applied to I in polynomial time, or it can be concluded in polynomial time that I is a YES-instance.*

Proof sketch. It can easily be shown that an instance in which at least k agents have an internal set is a YES-instance. Else, if the instance contains at least $k \cdot f_d(k)$ internal sets, we can use Lemma 2.2 to show that Reduction rule 1 can be applied in polynomial time. Otherwise, the number of internal sets is bounded, and we show that Reduction rule 2 can be applied. In that case, the number of petals in a sunflower that are intersected by internal sets is also bounded and the core of a sunflower is intersected by internal sets from at most d agents. Since $|\mathcal{S}| \geq g_d(k)$, Lemma 2.2 yields a sufficiently large sunflower such that some of its sets have the same number of elements from each agent with an internal set intersecting the core. Some subset of this is a sunflower meeting the prerequisites from Reduction rule 2 and this can be found in polynomial time. ◀

Now, the above insights can be used to give the kernel. Given an RP-d-SP instance I, it starts by greedily computing an inclusion-wise maximal packing $\mathcal{X}$ of disjoint sets. In the full version, we show that I is a YES-instance if $|\mathcal{X}| > k$. Otherwise, the union of all sets in $\mathcal{X}$ forms a hitting set of size $\leq k \cdot d$, meaning that Reduction rules 1 and 2 are safe to apply. Lemma 3.5 shows that at least one of them is applicable if $|\mathcal{X}| \geq g_d(k)$ and we repeatedly apply one until an instance is obtained with at most $g_d(k)$ sets. This proves the following.

▶ **Theorem 3.6 (★).** *There is a polynomial-time algorithm that takes an RP-d-SP instance as input and either correctly determines that this is a YES-instance or returns an equivalent RP-d-SP instance with at most $g_d(k)$ sets and a hitting set of size at most $k \cdot d$. Hence, for constant d, RP-d-SP admits a kernel of size $g_d(k) = \mathcal{O}\left(\left(k^{d+1}\right)^d\right)$.*

This kernel can be used as a subroutine to solve RP-d-SP. Since it outputs an equivalent instance with a bounded hitting set, the number of collections of pairwise disjoint sets is also bounded. This limits the number of candidate solutions and alternative packings that a brute-force algorithm needs to consider, and it allows us to prove the following statement.

▶ **Theorem 3.7 (★).** *RP-d-SP is solvable in $2^{\mathcal{O}(k \log k)} + (n+m)^{\mathcal{O}(1)}$ time for each constant d.*

4 A tight ETH lower bound for rejection-proof kidney exchange

We prove that the algorithm presented in Theorem 3.7 is asymptotically optimal to solve RP-d-SP under the ETH for any $d \geq 3$. More strongly, we show that no faster algorithm exists for the problem RPKE-3 under the ETH, even when the input contains only two agents. To see that this is a stronger result, note that when $d \geq 3$ the RP-d-SP problem strictly generalizes RPKE-3. The construction also yields a lower bound for c-RPKE-3 for every $c \geq 2$. We use the ETH lower bound on Subgraph Isomorphism from Lemma 2.5 as starting point.

▶ **Lemma 4.1.** *Assuming the ETH, there is no integer $c \geq 2$ such that c-RPKE-3 is solvable in $2^{o(n \log n)}$ time, even when the input contains only two agents. The same lower bound applies to RPKE-3 with two agents.*

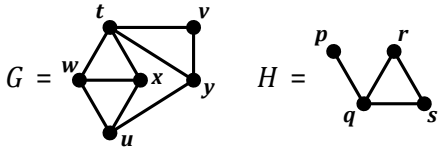
Proof. Let $c \geq 2$ be arbitrary. We prove the statement by a reduction from Subgraph Isomorphism. The resulting instance can serve as an input to both RPKE-3 and c-RPKE-3. First, we give the reduction after which we prove it is correct for both problems. A visual example of the reduction can be seen in Figure 2.

Let (G, H) be an input for the Subgraph Isomorphism problem consisting of two graphs and let $n_G := |V(G)|$ and $n_H := |V(H)|$. We assume that $n_H \leq n_G$, as (G, H) would trivially be a NO-instance otherwise. Now, we give our reduction by showing how to modify the input (G, H) to an input (G', V_1', V_2', k) with two agents for RPKE-3 and c-RPKE-3.

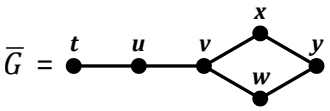
We start by defining the vertex set of G'. To define the complete vertex set, it is useful to think of it as six separate parts: five sets W_1, W_2, W_3, W_4, and W_5, and a separate vertex z. First, for every vertex $u \in V(H)$, we add a vertex u_1 to W_1 and a vertex u_2 to W_2. Next, for every vertex $x \in V(G)$, we add a vertex x_3 to W_3. Then, for every integer $i \in [n_G - n_H]$ we add a vertex i_4 to W_4 and a vertex i_5 to W_5. Finally, we add the additional vertex z to the graph. The vertex set is partitioned over the two agents by defining $V_1' := W_1 \cup W_3 \cup W_5 \cup \{z\}$ and $V_2' := W_2 \cup W_4$. Remark that $|W_1 \cup W_4| = |W_2 \cup W_5| = |W_3| = n_G$, so that $|V(G')| = 3n_G + 1$.

Next, we define the edge set of G'. Although G and H are undirected graphs, recall that G', being part of a kidney exchange instance, is a directed graph.

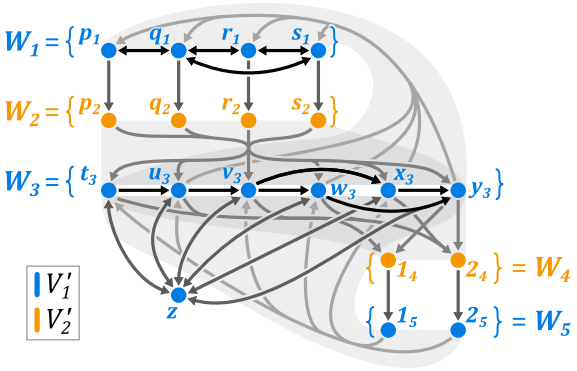
1. For every $\{u, v\} \in E(H)$, we add (u_1, v_1) and (v_1, u_1) to $E(G')$.
2. For every $u \in V(H)$, we add (u_1, u_2) to $E(G')$.
3. For every $u \in V(H)$ and every $x \in V(G)$, we add (u_2, x_3) and (x_3, u_1) to $E(G')$.
4. For every $i \in [n_G - n_H]$, we add (i_4, i_5) to $E(G')$.
5. For every $x \in V(G)$ and every $i \in [n_G - n_H]$, we add (x_3, i_4) and (i_5, x_3) to $E(G')$.
6. For every $x \in V(G)$, we add (x_3, z) and (z, x_3) to $E(G')$.
7. We fix an arbitrary ordering $\prec$ on $V(G)$. Then, for every $x, y \in V(G)$ such that $\{x, y\} \notin E(G)$ and $x \prec y$, we add (x_3, y_3) to $E(G')$.



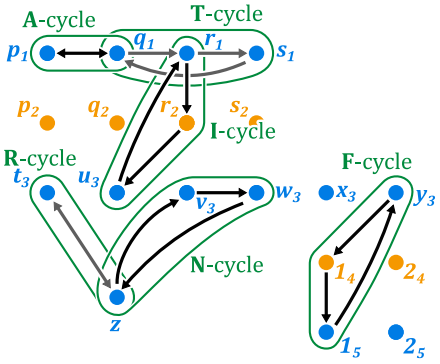
(a) An instance (G, H) of Subgraph Isomorphism as example input for the reduction.



(b) The complement of G, which is implicitly used to construct G'.



(c) Resulting graph G' obtained by running the reduction on the example input shown in Figure 2a.



(d) A simplified representation of G' indicating each of the different classes of cycles.

Figure 2 The reduction applied to an example input. For the sake of visualization, not all edges in the graph G' are shown. Four (partially overlapping) areas shaded in gray indicate that all vertices of W_2 and W_5 each have outgoing edges to all vertices in W_3 and that all vertices of W_3 have outgoing edges to all vertices of W_1 and W_4. Vertex colors are used to indicate the partition of the vertex set over the two agents with blue vertices belonging to V_1' and orange vertices to V_2'.

Finally, we define $k = 3 \cdot n_G$. Note that this is one less than the total number of vertices in G', denoted as $n := |V(G')| = 3n_G + 1$. This concludes the description of the reduction.

We distinguish six different classes of 3-cycles that live in G'. Below, we describe for each of these classes what their intuitive role is in the reduction and we indicate each class of cycles with a letter to refer to them more easily later in the proof. See also Figure 2d.

- For every edge $\{u, v\} \in E(H)$, the graph G' contains the cycle $\langle u_1, v_1 \rangle$. We call such cycles A-cycles for encoding the **a**djacencies in H.
- For every triangle in H on vertices u, v, and w, the graph G' contains both orientations of the directed cycle on vertices u_1, v_1 and w_1. We call such cycles T-cycles due to their correspondence to **t**riangles in the graph H. These cycles originate merely as a byproduct of constructing the A-cycles and serve no explicit purpose in our reduction.
- For every $u \in V(H)$ and every $x \in V(G)$, the graph G' contains the cycle $\langle u_1, u_2, x_3 \rangle$. We call such cycles I-cycles, as they are used to encode subgraph **i**somorphisms from H to G: including a cycle $\langle u_1, u_2, x_3 \rangle$ will correspond to mapping $u \in V(H)$ to $x \in V(G)$ in a subgraph isomorphism. See also Figure 3 (left image) for an example.
- For every $x \in V(G)$ and every $i \in [n_G - n_H]$, the graph G' contains the cycle $\langle x_3, i_4, i_5 \rangle$. We call such cycles F-cycles as they are used in the construction of optimal solutions as **f**iller cycles to cover vertices in W_3 that are not already covered by I-cycles.
- For every $x, y \in V(G)$ with $\{x, y\} \notin E(G)$, the graph G' contains the cycle $\langle x_3, y_3, z \rangle$ (or $\langle y_3, x_3, z \rangle$ depending on whether $x \prec y$). We call such cycles N-cycles for encoding the **n**on-adjacencies in G.

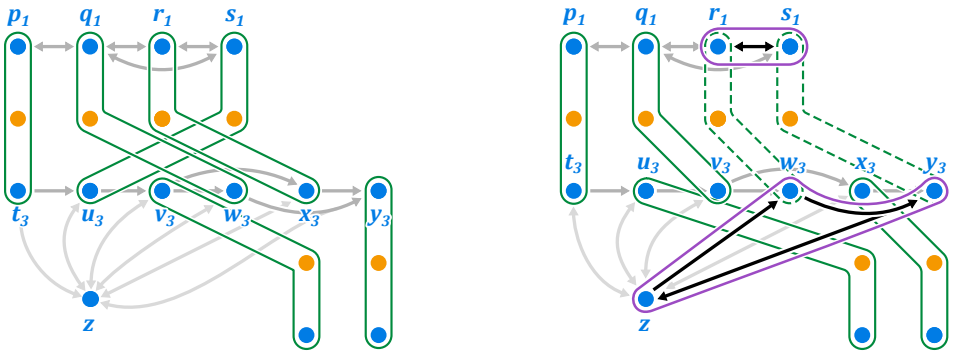


Figure 3 Both figures depict the same graph G' as in Figure 2, each with a different packing of cycles shown in green. For readability, only some edges are shown. The cycle packing on the left encodes the subgraph isomorphism from H to G that maps p to t, q to w, r to x, and s to u. The green packing on the right encodes a function that maps r to w and s to y. This is not a subgraph isomorphism because $\{r, s\} \in E(H)$ and $\{w, y\} \notin E(G)$. Hence, agent 1 can reject the green packing by proposing an alternative one in which the dashed cycles are replaced by the purple cycles.

- Finally, for every $x \in V(G)$, the graph G' contains the cycle $\langle x_3, z \rangle$. Much like T-cycles, these cycles serve no explicit purpose in our reduction and they originate merely as byproduct of constructing the N-cycles. We call them R-cycles, since optimal rejection-proof solutions typically do not include any of them, making these cycles mostly **r**edundant.

Before proving the correctness of our reduction, observe that G' does not contain any 3-cycles other than those contained in the six classes described above. In particular, note that $G'[W_3]$ is a DAG as the obvious extension of $\prec$ to W_3 forms a topological ordering. Finally, cycles whose length is greater than 3 will not be relevant in our proof, since RPKE-3 and c-RPKE-3 only deal with 3-cycles by definition.

Most of the remainder of this proof is used to show the correctness of the reduction. We do this by proving that H is isomorphic to a subgraph of G if and only if G' has a (c-)rejection-proof packing of 3-cycles that covers at least k vertices. Below, we prove the two directions of this bi-implication separately.

▷ Claim 4.2. If G' contains a 2-rejection-proof packing of 3-cycles that covers at least $k = 3 \cdot n_G$ vertices (e.g. because this packing is even (completely) rejection-proof), then H is isomorphic to a subgraph of G.

Proof. Let $\mathcal{C}$ be a 2-rejection-proof packing of 3-cycles in G' that covers at least $k = 3 \cdot n_G$ vertices. Before showing how to construct from $\mathcal{C}$ a subgraph isomorphism from H to G, we derive some structural properties of $\mathcal{C}$. We start by observing that at most one vertex from G' is not covered by $\mathcal{C}$. We use this property to show that $\mathcal{C}$ must contain n_H I-cycles.

First, we argue that the number of I-cycles in $\mathcal{C}$ must be more than $n_H - 2$. Since vertices from W_2 only appear in I-cycles and each I-cycle contains exactly one of these vertices, at least two vertices from W_2 would be uncovered by $\mathcal{C}$ otherwise. This would contradict the assumption that $\mathcal{C}$ leaves at most one vertex uncovered.

Next, we argue that the number of I-cycles in $\mathcal{C}$ cannot be exactly $n_H - 1$. In that case exactly one vertex u_1 from W_1 and exactly one vertex v_2 from W_2 would not be covered by an I-cycle in $\mathcal{C}$. If v_2 is not covered by an I-cycle in $\mathcal{C}$, this means that it is not covered at all by $\mathcal{C}$. Furthermore, u_1 only appears in I-cycles, A-cycles, and T-cycles. By assumption, it is

not covered by an I-cycle in $\mathcal{C}$. Moreover, it cannot be covered by an A-cycle or T-cycle in $\mathcal{C}$ either: such cycles contain at least two vertices from W_1, but u_1 is the only vertex in W_1 that is not covered by an I-cycle in $\mathcal{C}$. As such, $\mathcal{C}$ does not contain A-cycles or T-cycles since these would not be vertex-disjoint from all I-cycles in $\mathcal{C}$. This means that neither v_2 nor u_1 is covered by $\mathcal{C}$, contradicting that $\mathcal{C}$ leaves at most one vertex uncovered.

By arguing that the number of I-cycles in $\mathcal{C}$ must be strictly more than $n_H - 2$ but not exactly $n_H - 1$, we have shown that the number of I-cycles in $\mathcal{C}$ is exactly n_H. This means that all vertices from W_1 and W_2 are covered by I-cycles in $\mathcal{C}$.

Next, we see that $\mathcal{C}$ must include $n_G - n_H$ F-cycles. If it were to contain less F-cycles, at least one vertex from W_4 and W_5 each would not be covered by F-cycles. Since vertices from these two sets appear only in F-cycles, this would mean that they are not covered by $\mathcal{C}$ at all. In turn, this would contradict the observation that leaves $\mathcal{C}$ at most one vertex uncovered.

By including n_H I-cycles and $(n_G - n_H)$ F-cycles, all vertices of G' except z are covered. Since every cycle contains a vertex different from z, the packing $\mathcal{C}$ does not contain any other cycles. Knowing the structure of $\mathcal{C}$ allows us to construct a subgraph isomorphism $\varphi : V(H) \to V(G)$.

Let $u \in V(H)$ be arbitrary, and let $C_u \in \mathcal{C}$ be the unique I-cycle from $\mathcal{C}$ that contains u_1. Since every vertex from W_1 is contained in exactly one I-cycle in $\mathcal{C}$, this is well-defined. Let $x_3 \in W_3$ be the unique vertex from W_3 in C_u. Then, we define $\varphi(u) = x$. To complete the construction of φ, we do the same for every $u \in V(H)$.

This construction yields a function φ that maps every vertex in H to a unique vertex in G. To show that φ is in particular a subgraph isomorphism, it remains to show that for every $\{u, v\} \in E(H)$ it holds that $\{\varphi(u), \varphi(v)\} \in E(G)$. Suppose for contradiction that there is some $\{u, v\} \in E(H)$ for which this is not the case, i.e. $\{\varphi(u), \varphi(v)\} \notin E(G)$. We show that agent 1 would then reject $\mathcal{C}$. The right image in Figure 3 shows an example of this situation.

Let $x = \varphi(u)$ and $y = \varphi(v)$. Assume w.l.o.g. that $x \prec y$ in the arbitrary ordering fixed onto $V(G)$ in step 7 of the construction of $E(G')$. Then the following cycles exist:

- Because $\{u, v\} \in E(H)$, the A-cycle $\langle u_1, v_1 \rangle$ is added to G' in construction step 1.
- Because $\{x, y\} \notin E(G)$, the N-cycle $\langle x_3, y_3, z \rangle$ is added to G' in steps 6 and 7.

Now, agent 1 can reject the solution by proposing an alternative packing to $\mathcal{C}$ in which $\langle u_1, u_2, x_3 \rangle$ and $\langle v_1, v_2, y_3 \rangle$ are removed and the cycles $\langle u_1, v_1 \rangle$ and $\langle x_3, y_3, z \rangle$ are added.

This modification to $\mathcal{C}$ yields a packing of cycles that are still pairwise disjoint and which cover all vertices of V_1'. Moreover, the two cycles added by agent 1 are indeed 1-internal cycles in the graph, which means that agent 1 will indeed reject $\mathcal{C}$ whenever there is a $\{u, v\} \in E(H)$ such that $\{\varphi(u), \varphi(v)\} \notin E(G)$. This contradicts the assumption that $\mathcal{C}$ is 2-rejection-proof, so we conclude that φ is a valid subgraph isomorphism from H to G. ◁

▷ Claim 4.3 (★). If H is isomorphic to a subgraph of G, then G' contains a rejection-proof packing of 3-cycles (which is thus also c-rejection-proof) that covers $k = 3 \cdot n_G$ vertices.

Proof sketch. Let $\varphi : V(H) \to V(G)$ be a subgraph isomorphism from H to G. To encode this isomorphism as a packing of 3-cycles, we include the I-cycle $\langle u_1, u_2, x_3 \rangle$ for every $u \in V(H)$ and $x \in V(G)$ such that $x = \varphi(u)$. We extend this to a packing $\mathcal{C}$ that covers k vertices by adding arbitrary but disjoint F-cycles that cover the $n_G - n_H$ vertices from W_3 that are not contained in any of the added I-cycles. It remains to show that this packing is rejection-proof.

As $\mathcal{C}$ leaves only the vertex z uncovered, which belongs to V_1', only agent 1 can reject and must do so by constructing an alternative packing that covers all vertices of V_1'. In particular, they must propose to add some internal cycle C_z that includes z. By construction of G', this cycle is either an R-cycle or an N-cycle and we can show with a case distinction that neither of those two options extends to a valid rejection for agent 1. ◁

Finally, we argue that Claims 4.2 and 4.3 combine to prove the original lemma statement. Together, they prove the correctness of the reduction from Subgraph Isomorphism to both RPKE-3 and c-RPKE-3. Moreover, the reduction can be executed in polynomial time. Hence, if an algorithm were to exist that solves either RPKE-3 or c-RPKE-3 in $2^{o(n \log n)}$ time, this algorithm could be used to determine in $2^{o(n \log n)} = 2^{o(n_G \log n_G)}$ time whether H is isomorphic to a subgraph of G. By Lemma 2.5, this is not possible assuming the ETH. ◀

For completeness, we remark without proof that the construction above can be modified to prove that for any $d \geq 3$ and $c \geq 3$, an algorithm that solves c-RPKE-d in time $2^{o(n \log n)}$ would violate the ETH.

5 A single-exponential algorithm for 1-rejections

In the previous section, we have shown that under the ETH it is impossible to solve RPKE-d (or its restricted case 2-RPKE-d) in time $2^{o(n \log n)}$. This raises the question whether a single-exponential running time is possible for 1-RPKE-d. We answer this question positively by providing a $3^n \cdot (n+m)^{\mathcal{O}(1)}$ algorithm for the generalization 1-RP-d-SP.

▶ **Theorem 5.1 (★).** *For each fixed d, the 1-RP-d-SP problem can be solved in $3^n \cdot (n+m)^{\mathcal{O}(1)}$ time.*

Proof sketch. The main idea for solving an instance $I = (U_1, \ldots, U_p, \mathcal{S}, k)$ of 1-RP-d-SP is as follows. Let $U := \bigcup_{i \in [p]} U_i$ be the complete universe. The algorithm iterates over all subsets $U' \subseteq U$ of size at least k, each of which could be the set of elements covered by a solution to I. It then tries to find a solution covering exactly U'. The main insight now is the following: whether or not a set $A \in \mathcal{S}[U']$ leads to a 1-rejection by agent i in a solution that covers exactly U', is uniquely determined by the combination of U' and A. It is independent of the identities of the other sets used in the solution. Agent i performs a 1-rejection for A if and only if there is a collection of pairwise disjoint i-internal sets avoiding $U' \setminus A$ that covers more elements from U_i than A does. For each choice of U' we can therefore compute a collection $\mathcal{S}'$ of sets that are feasible to be used in a 1-rejection-proof solution covering exactly U'. Testing whether there is a 1-rejection-proof solution covering exactly U' then turns into an instance of Exact Set Cover over a universe of size $|U'| \leq |U|$, which can be solved efficiently via Lemma 2.1. The exact running time follows from some small optimizations and the binomial theorem. ◀

6 Conclusion and discussion

In this paper we started the parameterized complexity analysis of rejection-proof kidney exchange, which was recently introduced by Blom et al. [6] due to practical applications. Our FPT algorithm shows that the complexity of the problem is governed by the size of the solution, rather than the size of the instance. For the single-agent version of the kidney exchange problem, its parameterized complexity has also been considered with respect to structural parameterizations such as treewidth. We leave such investigations of rejection-proof kidney exchange to further work. Looking beyond the kidney exchange (i.e., d-cycle packing) problem studied in this paper, one could consider rejection-proof versions of other packing problems such as Chordless Cycle Packing [21] and Odd Cycle Packing [18].

References

1 David J. Abraham, Avrim Blum, and Tuomas Sandholm. Clearing algorithms for barter exchange markets: enabling nationwide kidney exchanges. In *Proceedings of the 8th ACM conference on Electronic commerce*, EC07. ACM, June 2007. doi:10.1145/1250910.1250954.

2 Nikhil Agarwal, Itai Ashlagi, Eduardo Azevedo, Clayton R. Featherstone, and Ömer Karaduman. Market failure in kidney exchange. *American Economic Review*, 109(11):4026–4070, November 2019. doi:10.1257/aer.20180771.

3 Itai Ashlagi, Felix Fischer, Ian A. Kash, and Ariel D. Procaccia. Mix and match: A strategyproof mechanism for multi-hospital kidney exchange. *Games and Economic Behavior*, 91:284–296, May 2015. doi:10.1016/j.geb.2013.05.008.

4 Peter Biro, Marton Gyetvai, Xenia Klimentova, Joao Pedro Pedroso, William Pettersson, and Ana Viana. Compensation scheme with shapley value for multi-country kidney exchange programmes. In *ECMS 2020 Proceedings edited by Mike Steglich, Christian Mueller, Gaby Neumann, Mathias Walther*, pages 129–136. ECMS, June 2020. doi:10.7148/2020-0129.

5 Péter Biró, Joris van de Klundert, David Manlove, William Pettersson, Tommy Andersson, Lisa Burnapp, Pavel Chromy, Pablo Delgado, Piotr Dworczak, Bernadette Haase, Aline Hemke, Rachel Johnson, Xenia Klimentova, Dirk Kuypers, Alessandro Nanni Costa, Bart Smeulders, Frits Spieksma, María O. Valentín, and Ana Viana. Modelling and optimisation in european kidney exchange programmes. *European Journal of Operational Research*, 291(2):447–456, June 2021. doi:10.1016/j.ejor.2019.09.006.

6 Danny Blom, Bart Smeulders, and Frits Spieksma. Rejection-proof mechanisms for multi-agent kidney exchange. *Games and Economic Behavior*, 143:25–50, January 2024. doi:10.1016/j.geb.2023.10.015.

7 Patrizia Burra and Manuela De Bona. Quality of life following organ transplantation. *Transplant International*, 20(5):397–409, May 2007. doi:10.1111/j.1432-2277.2006.00440.x.

8 Margarida Carvalho and Andrea Lodi. A theoretical and computational equilibria analysis of a multi-player kidney exchange program. *European Journal of Operational Research*, 305(1):373–385, February 2023. doi:10.1016/j.ejor.2022.05.027.

9 Margarida Carvalho, Andrea Lodi, João Pedro Pedroso, and Ana Viana. Nash equilibria in the two-player kidney exchange game. *Mathematical Programming*, 161(1–2):389–417, April 2016. doi:10.1007/s10107-016-1013-7.

10 Marek Cygan, Fedor V. Fomin, Alexander Golovnev, Alexander S. Kulikov, Ivan Mihajlin, Jakub Pachocki, and Arkadiusz Socala. Tight bounds for graph homomorphism and subgraph isomorphism. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1643–1649. SIAM, 2016. doi:10.1137/1.9781611974331.CH112.

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 P. Erdös and R. Rado. Intersection theorems for systems of sets. *Journal of the London Mathematical Society*, s1-35(1):85–90, January 1960. doi:10.1112/jlms/s1-35.1.85.

13 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer Berlin, Heidelberg, 2006. doi:10.1007/3-540-29953-X.

14 Florent Foucaud, Esther Galby, Liana Khazaliya, Shaohua Li, Fionn Mc Inerney, Roohani Sharma, and Prafullkumar Tale. Metric dimension and geodetic set parameterized by vertex cover. In *42nd International Symposium on Theoretical Aspects of Computer Science, STACS 2025, March 4-7, 2025, Jena, Germany*, volume 327 of *LIPIcs*, pages 33:1–33:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPICS.STACS.2025.33.

15 Chen Hajaj, John Dickerson, Avinatan Hassidim, Tuomas Sandholm, and David Sarne. Strategy-proof and efficient kidney exchange using a credit mechanism. *Proceedings of the AAAI Conference on Artificial Intelligence*, 29(1), February 2015. doi:10.1609/aaai.v29i1.9322.

16 Úrsula Hébert-Johnson, Daniel Lokshtanov, Chinmay Sonar, and Vaishali Surianarayanan. Parameterized complexity of kidney exchange revisited. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI 2024, Jeju, South Korea, August 3-9, 2024*, pages 76–84. ijcai.org, 2024. doi:10.24963/ijcai.2024/9.

17 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.

18 Ken-ichi Kawarabayashi and Bruce A. Reed. Odd cycle packing. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 695–704. ACM, 2010. doi:10.1145/1806689.1806785.

19 Xenia Klimentova, Ana Viana, João Pedro Pedroso, and Nicolau Santos. Fairness models for multi-agent kidney exchange programmes. *Omega*, 102:102333, July 2021. doi:10.1016/j.omega.2020.102333.

20 Arnab Maiti and Palash Dey. Parameterized algorithms for kidney exchange. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*, pages 405–411. ijcai.org, 2022. doi:10.24963/IJCAI.2022/58.

21 Dániel Marx. Chordless cycle packing is fixed-parameter tractable. In *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPIcs*, pages 71:1–71:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.ESA.2020.71.

Parameterized Complexity of Vehicle Routing

Michelle Döring
Hasso Plattner Institute,
University of Potsdam, Germany

Jan Fehse
Hasso Plattner Institute,
University of Potsdam, Germany

Tobias Friedrich
Hasso Plattner Institute,
University of Potsdam, Germany

Paula Marten
Hasso Plattner Institute,
University of Potsdam, Germany

Niklas Mohrin
Hasso Plattner Institute,
University of Potsdam, Germany

Kirill Simonov
Department of Informatics,
University of Bergen, Norway

Farehe Soheil
Hasso Plattner Institute,
University of Potsdam, Germany

Jakob Timm
Hasso Plattner Institute,
University of Potsdam, Germany

Shaily Verma
Hasso Plattner Institute,
University of Potsdam, Germany

Abstract

The Vehicle Routing Problem (VRP) is a popular generalization of the Traveling Salesperson Problem. Instead of one salesperson traversing the entire weighted, undirected graph G, there are k vehicles available to jointly cover the set of clients $C \subseteq V(G)$. Every vehicle must start at one of the depot vertices $D \subseteq V(G)$ and return to its start. Capacitated Vehicle Routing (CVRP) additionally restricts the route of each vehicle by limiting the number of clients it can cover, the distance it can travel, or both.

In this work, we study the complexity of VRP and the three variants of CVRP for several parameterizations, in particular focusing on the treewidth of G. We present an FPT algorithm for VRP parameterized by treewidth. For CVRP, we prove paraNP- and W[·]-hardness for various parameterizations, including treewidth, thereby rendering the existence of FPT algorithms unlikely. In turn, we provide an XP algorithm for CVRP when parameterized by both treewidth and the vehicle capacity.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases Vehicle Routing Problem, Treewidth, Parameterized Complexity

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.10

Related Version *Full Version*: `https://arxiv.org/abs/2509.10361` [8]

Funding *Michelle Döring*: German Federal Ministry for Education and Research (BMBF) through the project "KI Servicezentrum Berlin Brandenburg" (01IS22092).

1 Introduction

Optimizing logistics has been an important driver of the theory of computing since its very dawn, and, with the ever-growing digitalization and decentralization of services, it remains a productive direction to this day. One of the canonical challenges in the area, the Vehicle Routing Problem (VRP), introduced by George Dantzig and John Ramser in 1959 [6], can be informally summarized as follows: "What is the optimal set of routes for a fleet of vehicles to traverse in order to deliver to a given set of customers?"

© Michelle Döring, Jan Fehse, Tobias Friedrich, Paula Marten, Niklas Mohrin, Kirill Simonov, Farehe Soheil, Jakob Timm, and Shaily Verma;
licensed under Creative Commons License CC-BY 4.0

To model this question, it is natural to use a weighted graph to represent the relevant locations and the travel costs between them, with particular vertices marked as the depots, where the vehicles and the goods are originally located, and with some other vertices marked as the clients that await the delivery. Formally, we consider VRP to be the following computational problem: Given an edge-weighted undirected graph (G, w), subsets $C, D \subseteq V(G)$, and integers k, r, determine whether there exists a collection of at most k closed walks in G, such that each walk starts and ends at a vertex of D, the walks cover all vertices in C, and their total weight is at most r.

Defined in this way, VRP also becomes interesting from the theoretical point of view, as it generalizes several important NP-hard problems. Notably, the famous Traveling Salesperson Problem is a special case of VRP with one vehicle and every vertex being a client, i.e., $k = 1$, $C = D = V(G)$. On the other hand, since VRP allows for multiple vehicles, it also captures packing and covering problems on graphs. For example, the Maximum Cycle Cover problem, where the task is to find a collection of k vertex-disjoint cycles in the given graph G that covers all vertices [5], is modeled by VRP with $C = D = V(G)$ and $r = |V(G)|$.

Based on the immediate hardness of the problem in the classical sense, practical solvers for vehicle routing are mostly based on heuristics, which provide guarantees on neither the quality of the solution nor on the running time. We refer to surveys by Braekers et al. [4] and by Mor and Speranza [11] for an in-depth introduction to the vast area of practical research on solving the vehicle routing problem, as well as for listing various variants of the vehicle routing problem that received practical interest. Stemming from numerous research works and the efficiency of the developed solvers, the vehicle routing problem itself sees applications to other domains, for example, routing traffic in computer networks [1].

On the other hand, theoretical results regarding VRP have been much scarcer, especially with respect to solving the problem exactly. The immediate NP-hardness of the problem together with the diversity of explicit (such as number of vehicles and tour lengths) and implicit parameters (such as treewidth that may be small in certain applications [1]) makes VRP a natural target for parameterized complexity. However, to the best of our knowledge, this avenue was largely unexplored until now, although TSP itself [3, 7] and other related problems such as cycle packing [2] are well-studied in the area.

Our contribution. In this work, we aim to close this gap and initiate the parameterized complexity study of VRP and its variants. First, based on the different variants of vehicle routing found in the literature [4, 11], we define a general vehicle routing setting GVRS that provides a unified interface to the individual problems. We then focus on two important special cases: the VRP problem, as defined above, and its capacitated variant, where each route is additionally subject to a size constraint. Specifically, in LoadCVRP, the input also contains the parameter ℓ, and each route in the solution can only serve at most ℓ clients. This is motivated by real-life constraints, such as the limited cargo space of vehicles and the limited working hours of drivers. On the other hand, the capacity constraint allows to encapsulate even more fundamental theoretical problems within the vehicle routing framework, such as Triangle Packing, corresponding to $\ell = 3$, and ℓ-Cycle Packing. Similarly, we define GasCVRP, where there is a limit on the edge-weight of each route, and LoadGasCVRP, where both edge-weight and number of clients are limited. The problem classification is covered in detail in Section 2.1.

We then proceed to investigate the parameterized complexity of these problems with respect to the parameters such as the number of vehicles k, the number of depots $|D|$, the number of clients $|C|$, the total weight of the solution r, the treewidth of the graph tw, and the maximum load ℓ in case of LoadCVRP. Our findings are summarized in Table 1.

Table 1 Overview of our results for Vehicle Routing variants. We omit the results for GasCVRP and LoadGasCVRP that are analogous to the listed results for LoadCVRP.

VRP Variant	Parameter	Complexity	Reference
VRP	$\|D\| + k$	paraNP-hard	Theorem 3.1 (MetricTSP)
	$\|C\|$	FPT	Theorem 3.2
	r	FPT	Corollary 3.3 ($r \geq \|C\|$)
	tw	FPT	Theorem 3.9
LoadCVRP	ℓ	paraNP-hard	Theorem 4.12 (TrianglePacking)
	$\|C\|$	FPT	Theorem 4.1
	r	FPT if 0-edges disallowed	Corollary 4.3 ($r \geq \|C\|$)
	$k + \ell$	FPT	Theorem 4.4 ($k \cdot \ell \geq \|C\|$)
	$\text{tw} + \|D\|$	paraNP-hard	Theorem 4.8 (BinPacking)
	$\text{tw} + k + \|D\|$	W[1]-hard	Theorem 4.8 (BinPacking)
	$\text{tw} + \ell$	XP	Corollary 4.25
LoadGasCVRP	$\text{tw} + \ell + \|D\|$	paraNP-hard	Theorem 4.15 (N3DMatching)
	$\text{tw} + g + \|D\|$	paraNP-hard	Corollary 4.16 (N3DMatching)

We start in Section 3 with the uncapacitated case, i.e., the VRP problem. As this problem presents a generalization of TSP, we can rule out tractability even for a constant number of vehicles and depots in Theorem 3.1. In Theorem 3.2 we also quickly conclude that the number of clients is a sufficient parameter for an FPT algorithm. This result directly implies that deciding whether a routing of weight at most r exists can be done in FPT-time in r.

The main result of Section 3 deals with structural properties of our network, namely its treewidth. While there are many known FPT algorithms parameterized by treewidth for various graph problems, the VRP problem presents a unique combination of several challenging aspects. The solution is composed of arbitrarily many individual objects, i.e., walks, which may use edges and vertices multiple times and can have arbitrary length. Neither the number of walks in the solution, nor the length of the individual walk, nor the multiplicity of an edge or a vertex in a walk is bound by the treewidth. The closest known starting point on the approach to VRP is the FPT algorithm by Schierreich and Suchý [14] for the Waypoint Routing Problem parameterized by treewidth, which is effectively a single-vehicle case of VRP. While resolving some of the challenges outlined above, their algorithm does not readily allow for the incorporation of the partitioning aspect as well, i.e., to deal with multiple vehicles, over which the clients need to be partitioned. We generalize this approach further and obtain an FPT algorithm for VRP parameterized by treewidth (Theorem 3.9). In fact, the result we obtain holds for the more general Edge-Capacitated Vehicle Routing Problem (EVRP), which additionally allows to specify arbitrary capacities on the edges, so that each edge can be used by the solution at most the specified number of times. One of the key ingredients behind the algorithm is the combinatorial observation that allows us to restrict the number of times an edge can be used by the solution. Based on this, we introduce an equivalent reformulation of VRP that only considers Eulerian submultigraphs. This reformulation can, in turn, be solved efficiently via dynamic programming over the tree decomposition. Unfortunately, the operations in dynamic programming required for dealing with multiple walks extend beyond the established single-exponential-time machinery developed by Bodlaender et al. [3]. The running time that our algorithm achieves is thus

of the form $2^{\mathcal{O}(\mathrm{tw}\log \mathrm{tw})} \cdot n^{\mathcal{O}(1)}$; improving this to purely single-exponential running time remains a challenging open question. Nevertheless, this result completes the classification with respect to the outlined parameters.

We then focus on capacitated vehicle routing in Section 4. Recall that LoadCVRP, GasCVRP, and LoadGasCVRP result from adding to VRP the restriction on the maximum number of clients per vehicle ℓ, the maximum length of a tour g, or both simultaneously. As our work shows, these three versions largely behave in the same way with respect to the parameters in question; therefore we mainly focus on LoadCVRP in order to introduce our results.

The additional constraints allow us to model different packing problems as capacitated vehicle routing problems, and by showing such inclusions, we derive several intractability results. In Section 4.2, we show that LoadCVRP is NP-hard even for maximum load $\ell = 3$, by providing a reduction from TrianglePacking. More surprisingly, by reduction from BinPacking, we can also show that treewidth alone is not enough for fixed-parameter tractability in the presence of the capacity constraints, even when combined with the number of vehicles k and the number of depot vertices $|D|$.

Based on this hardness result, we move on to consider treewidth in combination with the capacity parameters. We show complete intractability for LoadGasCVRP, even when combining treewidth, the maximum load, and the number of depots as a parameter, using N3DMatching in Section 4.3. However, XP tractability can be achieved by combining treewidth and the capacity parameter in LoadCVRP and GasCVRP, as well as combining both capacity parameters with treewidth in LoadGasCVRP. These algorithms are the main results of Section 4. To obtain these, we first introduce a generalization of the BinPacking problem that acts as an interface to the capacitated vehicle routing problems, and prove tractability of this problem for a combination of parameters. Finally, in Section 4.4 we present a dynamic programming algorithm that solves LoadGasCVRP by modeling the optimal combination of partial solutions as instances of generalized BinPacking.

The proofs of some theorems are omitted and can be found in the full version of this paper [8]. These theorems are marked with ($\star$).

2 Preliminaries

Let $\mathbb{N} = \{0, 1, 2, \ldots\}$ be the set of natural numbers, including zero. For $n \in \mathbb{N}$, we abbreviate $[n] = \{1, 2, \ldots, n\}$ and $[n]_0 = \{0\} \cup [n]$. A *multiset* is an unordered collection of elements of some universe U, where each element can occur multiple times. Formally, we model multisets as functions mapping each possible element to the number of occurrences. Alternatively, we use set notations with modified brackets as in these examples: $[\![1, 2, 2, 3]\!]$ (the multiset containing 1 and 3 once, and 2 twice), $[\![\lfloor n/2 \rfloor \mid n \in \mathbb{N}]\!]$ (the multiset containing each natural number twice). The cardinality of a multiset A is denoted as $|A| = \sum_{u \in U} A(u)$. For some sub-universe $U' \subseteq U$, we denote the restriction of A to U' as $A_{\downarrow U'}$. For two multisets A, B over the same universe U, we denote by $A + B$ the multiset union defined by $(A + B)(u) = A(u) + B(u)$ for every $u \in U$. For $n \in \mathbb{N}$, we denote by $n \cdot A$ the multiset obtained by uniting n copies of A. Let G be an undirected (multi-)graph. By convention, we usually abbreviate $n = |V(G)|$. A *walk* p in a graph G is defined as a sequence of vertices and edges $(v_1, e_1, v_2, e_2, \ldots, e_{\ell-1}, v_\ell)$, where the sequence begins at a vertex and traverses edges of the graph, allowing for the repetition of both vertices and edges. The *length* of a walk is given by the number of edges it contains, counted with multiplicity. For simplicity, we may represent a walk of length $\ell - 1$ by the sequence of its vertices, i.e., $(v_1, v_2, \ldots, v_\ell)$. A walk is *closed* if the first and last visited vertices are equal, otherwise it is *open*.

A (multi-)graph G is called *Eulerian* if all vertices in G have even degree. Equivalently, G is Eulerian when it is the (multiset-)union of closed walks. Note that, unlike the classic (connected) notion of Eulerian graphs, this definition does not require G to be connected. Given an edge $e \in E(G)$ and number $k \in \mathbb{N}$, we denote by $G - k \cdot e$ the multigraph G with k occurrences of e removed.

Let $w\colon E(G) \to \mathbb{N}$ be a weight function for the edges of G. For a walk $p = (v_1, v_2, \ldots, v_\ell)$ in G we define the weight of p as $w(p) = \sum_{i=1}^{\ell-1} w(v_i, v_{i+1})$. For a submultigraph $H \subseteq G$ we define the weight of H as $w(H) = \sum_{e \in E(H)} w(e)$, where parallel edges in H should appear separately in the sum, that is, the weight of H accounts for the multiplicity of the edges.

Finally, we recall the definitions of tree decompositions, treewidth, and nice tree decompositions, which we will use for our treewidth-based algorithms.

▶ **Definition 2.1** (Tree Decomposition, [13, 5])**.** A *tree decomposition* of an undirected graph G is a pair $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ consisting of a tree T and for each vertex t of T a set of vertices $X_t \subseteq V(G)$, so that

- $\bigcup_{t \in V(T)} X_t = V(G)$,
- for all $\{u, v\} \in E(G)$, there is $t \in V(T)$ such that $u, v \in X_t$, and
- for all $v \in V(G)$, the vertices $t \in V(T)$ with $v \in X_t$ form a connected subtree of T.

The sets X_t are called *bags*. We abbreviate $t \in V(T)$ as $t \in \mathcal{T}$. The *width* of $\mathcal{T}$ is defined as $\text{width}(\mathcal{T}) = \max_{t \in \mathcal{T}} |X_t| - 1$ and the *treewidth* of G is the minimum width of any tree decomposition of G. ⌟

▶ **Definition 2.2** (Nice Tree Decomposition, [5])**.** A tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ is called *nice* if T is rooted at a vertex $r \in V(T)$ with $X_r = \emptyset$ and all vertices $t \in V(T)$ are of one of the following kinds:

- Leaf Node: t has no children and $X_t = \emptyset$.
- Introduce Vertex Node: t has exactly one child t' and there is a vertex $v \notin X_{t'}$ such that $X_t = X_{t'} \cup \{v\}$.
- Introduce Edge Node: t has exactly one child t', $X_t = X_{t'}$, and t is labeled with an edge $\{u, v\} \in E(G)$ with $u, v \in X_t$.
- Forget Node: t has exactly one child t' and there is a vertex $v \in X_{t'}$ such that $X_t = X_{t'} \setminus \{v\}$.
- Join Node: t has exactly two children t_1, t_2 and $X_t = X_{t_1} = X_{t_2}$.

For each edge $e \in E(G)$, there must be exactly one Introduce Edge Node labeled with e. If G is a multigraph, each copy of an edge is introduced separately.

For $t \in \mathcal{T}$, let S be the set of all descendants of t in T (including t itself). We denote the set of all introduced vertices up until t as $V_t^{\downarrow} = \bigcup_{t' \in S} X_{t'}$. Similarly, we define $E_t^{\downarrow}$ to be the (multi-)set of all introduced edges up until t and $G_t^{\downarrow} = (V_t^{\downarrow}, E_t^{\downarrow})$. ⌟

2.1 Classifying Vehicle Routing Problems

The literature deals with many variants of vehicle routing problems [4, 11]. To incorporate those in a single notion, we introduce the following *Generalized Vehicle Routing Setting*.

▶ **Definition 2.3** (Generalized Vehicle Routing Setting)**.** An instance of the *Generalized Vehicle Routing Setting* (GVRS) consists of a graph G with edge weights $w\colon E(G) \to \mathbb{N}$, a nonempty set of *depots* $D \subseteq V(G)$, a set of *clients* $C \subseteq V(G)$, an integer $k \in \mathbb{N}$ representing the number of vehicles, and a predicate Γ modeling additional constraints. We call a set of k' walks $R = \{p_1, p_2, \ldots, p_{k'}\}$ together with an assignment of vehicles $A\colon C \to [k']$ a *routing* for (G, w, D, C, k, Γ) if

1. $k' \leq k$ (the routing uses at most k vehicles),
2. $\forall p = (v_1, v_2, \ldots, v_\ell) \in R : v_1 \in D$ (every walk starts in one of the depots),
3. $\forall c \in C : c \in p_{A(c)}$ (every client is visited by its assigned vehicle), and
4. $\Gamma(R, A)$ (the additional constraints are satisfied).

The weight of a routing R is defined as $w(R) = \sum_{p \in R} w(p)$.

A problem instance may ask whether a feasible routing exists or whether there exists a routing R with $w(R) \leq r$, for some $r \in \mathbb{N}$. A routing of minimum weight is called *optimal*. ⌟

This paper studies four vehicle routing problems, denoted here as VRP, LoadCVRP, GasCVRP, and LoadGasCVRP for undirected graphs. The problem definitions will contain the additional constraint that each vehicle must return to the depot at which it started. More formally, we will use

$$\Gamma_{\text{closed}}(R, A) \iff \forall p \in R : p \text{ is closed.}$$

▶ **Definition 2.4** (VRP)**.** The *(Uncapacitated) Vehicle Routing Problem* (VRP) is a GVRS with an additional parameter $r \in \mathbb{N}$. It asks whether there exists a routing (R, A) with $w(R) \leq r$, subject to $\Gamma = \Gamma_{\text{closed}}$. ⌟

In Section 3, we obtain the main result for VRP, a parameterized algorithm for instances with bounded treewidth, by considering EVRP, a slightly generalized version of VRP.

▶ **Definition 2.5** (EVRP)**.** The *Edge-Capacitated Vehicle Routing Problem* (EVRP) is a GVRS with additional parameters $r \in \mathbb{N}$ and $\kappa \colon E(G) \to \mathbb{N}$. It asks whether there exists a routing (R, A) with $w(R) \leq r$, subject to $\Gamma = \Gamma_{\text{closed}} \wedge \Gamma_\kappa$, where

$$\Gamma_\kappa(R, A) \iff \forall e \in E : e \text{ is used at most } \kappa(e) \text{ times in } R.$$ ⌟

The literature on vehicle routing problems often considers *capacitated* variants [4, 11], usually denoted as CVRP. In contrast to EVRP, these capacities are placed on the vehicles individually. However, the term CVRP can refer to different kinds of capacities: *load* and *gas* capacities. In this paper, we consider both interpretations of CVRP.

In LoadCVRP, each client c has a demand for $\Lambda(c)$ units of some good, which must be delivered by its assigned vehicle. The vehicles, in turn, have a capacity ℓ on the *load* they can carry on their trip. Notably, this capacity ℓ is equal for all vehicles.

▶ **Definition 2.6** (LoadCVRP)**.** The *Load-Capacitated Vehicle Routing Problem* (LoadCVRP) is a GVRS with additional parameters $r \in \mathbb{N}$, $\ell \in \mathbb{N}$, and $\Lambda \colon C \to \mathbb{N}^+$. It asks whether there exists a routing (R, A) with $w(R) \leq r$, subject to $\Gamma = \Gamma_{\text{closed}} \wedge \Gamma_{\ell,\Lambda}$, where

$$\Gamma_{\ell,\Lambda}(R, A) \iff \forall i \in [|R|] : \sum_{c \in A^{-1}(i)} \Lambda(c) \leq \ell.$$ ⌟

A similar restriction is given in GasCVRP, where each route is limited by the amount g of *gas* (fuel) a vehicle can carry. Similar to LoadCVRP, the capacity g is equal for all vehicles.

▶ **Definition 2.7** (GasCVRP)**.** The *Gas-Capacitated Vehicle Routing Problem* (GasCVRP) is a GVRS with additional parameters $r \in \mathbb{N}$ and $g \in \mathbb{N}$. It asks whether there exists a routing (R, A) with $w(R) \leq r$, subject to $\Gamma = \Gamma_{\text{closed}} \wedge \Gamma_g$, where

$$\Gamma_g(R, A) \iff \forall p \in R : w(p) \leq g.$$ ⌟

Finally, LoadGasCVRP combines the constraints of LoadCVRP and GasCVRP.

▶ **Definition 2.8** (LoadGasCVRP)**.** The *Load-and-Gas-Capacitated Vehicle Routing Problem* (LoadGasCVRP) is a GVRS with additional parameters $r \in \mathbb{N}$, $\ell \in \mathbb{N}$, $\Lambda \colon C \to \mathbb{N}^+$, and $g \in \mathbb{N}$. It asks whether there exists a routing (R, A) with $w(R) \leq r$, subject to $\Gamma = \Gamma_{\text{closed}} \wedge \Gamma_{\ell,\Lambda} \wedge \Gamma_g$. ⌟

3 Uncapacitated Vehicle Routing

In this section, we establish the parameterized landscape around the Uncapacitated Vehicle Routing Problem. We show hardness of VRP when parameterized by the number of depots $|D|$ and vehicles k. In turn, we show that VRP is tractable when parameterized by the number of clients $|C|$. Finally, we present an algorithm which proves that EVRP (and thereby VRP) is tractable on graphs of bounded treewidth tw.

Note that for VRP routings, it suffices to find a set R of at most k closed walks that connect every client to a depot. This is because the predicate Γ does not pose any requirements on the assignment of vehicles A. In fact, an assignment A can be chosen based on R by arbitrarily selecting one of the vehicles which visit a given client. We thus do not mention the assignment A any further in this section.

Since MetricTSP is a special case of VRP where $C = V(G)$, $D = \{v\}$, and $k = 1$, we obtain the following result:

▶ **Theorem 3.1** (⋆). VRP *parameterized by* $|D| + k$ *is* paraNP*-hard.*

Note that the above observation still requires a large number of clients. In the next theorem, we see that we can efficiently solve VRP instances for constant numbers of clients.

▶ **Theorem 3.2** (⋆). *There is an algorithm that computes an optimal* VRP *routing in* $|C|^{\mathcal{O}(|C|)} \cdot n^{\mathcal{O}(1)}$ *steps.*

While the above algorithm outputs an optimal routing, it can also be used to solve the decision variant of VRP as defined in Definition 2.4. It also yields the following corollary:

▶ **Corollary 3.3** (⋆). *There is an algorithm that, given* $r \in \mathbb{N}$*, decides whether there exists a* VRP *routing of weight at most* r *in* $f(r) \cdot n^{\mathcal{O}(1)}$ *steps, for some computable function* f*.*

Next, we present an FPT algorithm for VRP parameterized by the treewidth. Our algorithm generalizes the prior work by Schierreich and Suchý [14] on the *Waypoint Routing Problem* (WRP). WRP can be formulated in terms of GVRS as follows:

▶ **Definition 3.4** (WRP). The *Waypoint Routing Problem* (WRP) is a GVRS with $k = 1$, $D = \{s\}$, a designated vertex $t \in V(G)$, and additional parameters $r \in \mathbb{N}$ and $\kappa\colon E(G) \to \mathbb{N}$. It asks whether there exists a routing R (consisting of a single walk, due to $k = 1$) with $w(R) \leq r$, subject to $\Gamma = \Gamma_t \wedge \Gamma_\kappa$, where

$$\Gamma_t(R, A) \iff \forall p \in R : p \text{ ends at } t,$$
$$\Gamma_\kappa(R, A) \iff \forall e \in E : e \text{ is used at most } \kappa(e) \text{ times in } R.$$

▶ **Theorem 3.5** ([14]). *There is an algorithm that computes an optimal* WRP *routing in* $2^{\mathcal{O}(\mathrm{tw})} \cdot n^{\mathcal{O}(1)}$ *steps, where* tw *denotes the treewidth of* G*.*

Schierreich and Suchý [14] obtain this result by reducing WRP to a normal form where $s = t$, effectively replacing the constraint Γ_t by Γ_{closed}. Next, they reformulate the problem to remove the parameter κ: Every edge $e \in E(G)$ is replaced by $\kappa(e)$ many parallel edges $e^1, e^2, \ldots, e^{\kappa(e)}$, each with weight $w(e)$. Given the obtained multigraph G', WRP now asks to find a connected, Eulerian subgraph $H \subseteq G$, which contains the depot s and all clients C.

Since WRP is very similar to a single-vehicle case of VRP, we aim to adapt the techniques from [14] to our work. In particular, we slightly generalize the VRP problem and consider EVRP that incorporates edge capacities (see Definition 2.5), making WRP a proper special case of our problem. The introduction of edge capacities κ to VRP yields the same reformulation to multigraphs as found for WRP.

▶ **Lemma 3.6** (⋆). *Let G' be the multigraph obtained by replacing all edges e of G by $\kappa(e)$ parallel edges as described above. There is an* EVRP *routing R of weight $r \in \mathbb{N}$ in G if and only if there is an Eulerian subgraph $H \subseteq G'$ of weight $w(H) = r$ with $C \subseteq V(H)$ and at most k connected components, so that every client $c \in C$ is reachable from a depot $d \in D$.*

Based on Lemma 3.6, we reuse the term *routing* to refer to Eulerian subgraphs $H \subseteq G'$ that connect every client to a depot and contain at most k connected components. Furthermore, we shift the goal of our algorithm to finding a suitable subgraph $H \subseteq G'$. Before we can move on from G and κ, we need to make one more simplification to the problem: Although replicating all edges e by $\kappa(e)$ copies greatly increases the size of the input graph, this increase can be avoided by the following observation, which can be proven easily.

▶ **Lemma 3.7.** *Let $H \subseteq G'$ be an* EVRP *routing and $e \in E(H)$ be an edge with multiplicity greater than 2. Then $H - 2e$ is an* EVRP *routing and $w(H - 2e) \leq w(H)$.*

By repeatedly applying Lemma 3.7, we see that it suffices to solve EVRP instances with $\kappa(e) \leq 2$ for all edges e. In this case, the introduction of $\kappa(e)$ copies of an edge e can only increase the input size by a factor of 2. Therefore, we will not mention the simple graph G and the edge capacities κ anymore. Instead, we work on multigraphs where each edge can only be used once.

It seems like the algorithm from Theorem 3.5 cannot be used directly to solve EVRP. However, its general approach still applies to vehicle routing problems. The main difference is that in WRP, all partial solutions eventually merge to a connected subgraph containing the source vertex s, whereas in EVRP, there are multiple connected components, each of which contains one of the depots. It turns out that this observation directly translates to an FPT-algorithm for EVRP parameterized by treewidth and the number of connected components.

▶ **Theorem 3.8** (⋆). *There is an algorithm that computes an optimal* EVRP *routing in $\binom{|D|}{d^*}(\mathrm{tw} + d^*)^{\mathcal{O}(\mathrm{tw}+d^*)} \cdot n^{\mathcal{O}(1)}$ steps where $d^* = \min\{|D|, k\}$.*

Importantly, the optimization employed by [14] to improve the running time from $\mathrm{tw}^{\mathcal{O}(\mathrm{tw})} \cdot n^{\mathcal{O}(1)}$ to $2^{\mathcal{O}(\mathrm{tw})} \cdot n^{\mathcal{O}(1)}$ cannot be used to improve Theorem 3.8. The optimization is based on the `reduce` operation from [3] that merges partial solutions which are equivalent when considering that they still have to be extended to the signature $\{\{s\}\}$ of a full solution. In Theorem 3.8, the signature at the root node is read from the partition $D[\emptyset]$, not $\{D\}$. Thus, a generalization of the setup by [3] to more complex partitions is needed to achieve a similar speedup.

While Theorem 3.8 yields that EVRP parameterized by $\mathrm{tw} + |D|$ is in FPT, it seems that the information tracked in the DP by [14] is almost sufficient for EVRP. This intuition is correct: By extending the weighted-partition setup of [3] and adapting the DP by [14], we manage to develop an algorithm for EVRP which is FPT when parameterized by just the treewidth of G. To adapt the setup we first introduce the coarsening relation and join operation on partitions. After presenting our extension of weighted partitions and adjusting the operations on weighted partitions by [3] to fit our definition, we can finally go on to present our tree decomposition based algorithm for EVRP. The complete list of definitions and adaptations can be found in the full version. We further show that the adapted operators correctly maintain and update each DP record, and using them, we derive the following theorem.

▶ **Theorem 3.9** (⋆). VRP *parameterized by* tw *is in* FPT.

4 Capacitated Vehicle Routing

In this section, we investigate the complexity of LoadCVRP, GasCVRP, and LoadGasCVRP. We observe that the three problems share most complexity characteristics. In particular, the requirement to adhere to vehicle capacities greatly increases the complexity. Whereas VRP admits an FPT algorithm when parameterized by the treewidth of G, all problems investigated in this section are NP-hard even on trees. We derive this hardness by encoding BinPacking instances in vehicle routing problems. In doing so, we also realize that the presence of zero-weight edges greatly impacts the tractability when parameterizing by the output weight r.

Next, we investigate the parameterized complexity with respect to the given vehicle capacities (ℓ for LoadCVRP and g for GasCVRP). We show paraNP-hardness by reduction from TrianglePacking for all three variants. This shows that capacitated vehicle routing problems do not only pose the challenge of "algebraic packing" as in BinPacking, but also one of "structural packing". We further provide paraNP-hardness results for LoadGasCVRP, when parameterized by treewidth, number of depots and either the load capacity or gas constraint, both by reduction from N3DMatching.

Finally, we find that the situation is not as dim when parameterizing by both the treewidth and the capacity value (for example $\mathrm{tw} + \ell$ for LoadCVRP). While we cannot yet answer whether or not there can exist an FPT algorithm for these parameters, we provide an XP algorithm for LoadGasCVRP parameterized by $\mathrm{tw} + \ell + g$, thereby ruling out paraNP-hardness. Additionally, our (to the best of our knowledge) novel FPT-algorithm for BinPacking parameterized by the capacity of each bin is a solid starting point for further analysis of the parameterized complexity of capacitated vehicle routing.

We start by extending some of our results from Section 3 to capacitated vehicle routing.

▶ **Theorem 4.1** ($\star$). *For each of* LoadCVRP, GasCVRP, *and* LoadGasCVRP, *there is an algorithm that computes an optimal routing in* $|C|^{\mathcal{O}(|C|)} \cdot n^{\mathcal{O}(1)}$ *steps.*

Note that the proof for Corollary 3.3 does not work for LoadCVRP in general as there is no sensible way to update the demand Λ when contracting zero-weight edges between two clients. This problem also occurs in LoadGasCVRP. Since client assignment is irrelevant for GasCVRP, and when no zero-weight edges are present the contraction step can be omitted, the proof of Corollary 3.3 applies analogously to the following statements.

▶ **Corollary 4.2.** *There is an algorithm that, given a* GasCVRP *instance and* $r \in \mathbb{N}$, *decides whether there exists a routing of weight at most* r *in* $f(r) \cdot n^{\mathcal{O}(1)}$ *steps, for some computable function* f.

▶ **Corollary 4.3.** *There is an algorithm that, given a* LoadCVRP *or* LoadGasCVRP *instance with no zero-weight edges and* $r \in \mathbb{N}$, *decides whether there exists a routing of weight at most* r *in* $f(r) \cdot n^{\mathcal{O}(1)}$ *steps, for some computable function* f.

Another insight is that limiting both k and the capacity constraint suffices for tractability, because the number of clients that can possibly be covered is limited.

▶ **Theorem 4.4** ($\star$). *All of the following problems are in* FPT*:*

- LoadCVRP *parameterized by* $k + \ell$,
- GasCVRP *parameterized by* $k + g$, *and*
- LoadGasCVRP *parameterized by* $k + \ell$.

4.1 Hardness from Bin Packing

Prior work regarding capacitated vehicle routing, such as by [12], has already observed the connection to BinPacking. Before we can formally capture the inclusion of BinPacking within vehicle routing, we first introduce the relevant prior work.

▶ **Definition 4.5** (BinPacking)**.** A BinPacking instance consists of a finite set U of items, with sizes given by $s\colon U \to \mathbb{N}^+$, a bin capacity $B \in \mathbb{N}^+$, and a number of bins $k \in \mathbb{N}$. It asks whether there exists a partition $U_1, U_2, \ldots, U_k$ of U such that for all $i \in [k]$ we have $\sum_{u \in U_i} s(u) \leq B$. Additionally, we define UnaryBinPacking as a BinPacking variant, in which all numerical inputs are encoded in unary. ⌟

Going forward, we assume that BinPacking instances do not contain items $u \in U$ with $s(u) > B$, as these instances can be rejected immediately.

▶ **Theorem 4.6** ([9])**.** *Both* BinPacking *and* UnaryBinPacking *are* NP*-complete.*

▶ **Theorem 4.7** ([10])**.** UnaryBinPacking *parameterized by the number of bins k is* W[1]*-hard.*

▶ **Theorem 4.8.** LoadCVRP*,* GasCVRP*, and* LoadGasCVRP *are* W[1]*-hard when parameterized by k, even with unit demands, unit weights, a single depot, and on trees.*

Proof. Similar to before, we give an extensive proof for LoadCVRP and briefly explain why it carries over to the other variants. Given a UnaryBinPacking instance (U, s, k, B), we construct a tree G with one central depot d and one branch for every item $u \in U$. To build such a branch, we add $s(u)$ vertices that form a path $v_u^1 v_u^2 \ldots v_u^{s(u)}$ and connect this path to the depot via an edge $\{d, v_u^1\}$. The set of clients is the set of all vertices on these paths, that is, $C = \{v_u^i \mid u \in U \wedge i \in [s(u)]\}$. We then output the LoadCVRP instance

$$(G, w_1, D = \{d\}, C, k, \Lambda_1, \ell = B, r = 2|E(G)|),$$

where w_1 and Λ_1 are the unit weight and unit demand functions $w_1\colon e \mapsto 1$ and $\Lambda_1\colon c \mapsto 1$ respectively. Figure 1 shows an example of the reduction. Note that even though we are creating $s(u)$ vertices for every $u \in U$, this reduction still runs in polynomial time, as we are reducing from UnaryBinPacking.

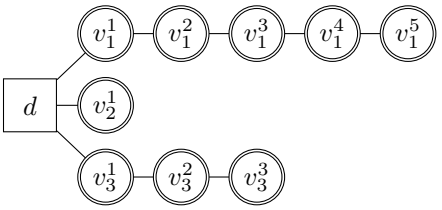


Figure 1 Example graph for $U = [3]$ with $s(1) = 5$, $s(2) = 1$, and $s(3) = 3$.

If there is a valid partition of U into k sets $U_1, \ldots, U_k$, which serves as a solution to the BinPacking instance, for every $i \in [k]$ we let a vehicle cover all clients v_u^j for $u \in U_i$ and $j \in [s(u)]$. This can be achieved by having the vehicle traverse the tree from the depot d to $v_u^{s(u)}$ and back for each $u \in U_i$. This way, every edge is walked exactly twice and a vehicle $i \in [k]$ covers $\sum_{u \in U_i} s(u) \leq B$ clients.

Suppose in turn that there is a LoadCVRP routing $(R = \{p_1, p_2, \ldots, p_{k'}\}, A)$ for the given instance. Note that as G is a tree, $r = 2|E(G)|$ is just enough so that the vehicles can reach the vertices $\{v_u^{s(u)} \mid u \in U\}$. Thus, the path of an item $u \in U$ is traversed by exactly one vehicle. Given that $\ell = B$, the partition given by $U_i = \{u \in U \mid v_u^1 \in V(p_i)\}$ for all $i \in [k']$ abides the constraints of BinPacking.

The same reduction works without demands for $g = 2B$, as any vehicle can then again visit at most B clients. For LoadGasCVRP the reduction works with $\ell = B$ and $g = 2B$. ◀

The above translates to the following result regarding treewidth:

▶ **Corollary 4.9.** LoadCVRP, GasCVRP, *and* LoadGasCVRP *are* paraNP*-hard when parameterized by* $\mathrm{tw} + |D|$ *and* W[1]*-hard when parameterized by* $\mathrm{tw} + k + |D|$*, even with unit demands and unit weights, where* tw *denotes the treewidth of* G*.*

When mapping BinPacking instances to vehicle routing, the number of bins corresponds to the number of vehicles, and the bin capacity corresponds to the capacity of each vehicle. As we see later in Corollary 4.20, BinPacking parameterized by B is in FPT. Thus, an FPT-reduction from BinPacking mapping a value from B to some vehicle capacity does not yield any hardness result. CVRP instances parameterized by the vehicle capacities could potentially be reduced to BinPacking parameterized by B, but such a reduction is unlikely to be applicable to the general case, as vehicle routing problems not only require a numerical partitioning as with BinPacking, but also a structural one found in TrianglePacking. Still, we return to this intuition in Section 4.4, where we develop an XP algorithm for LoadGasCVRP.

4.2 Hardness from Triangle Packing

In this section, we show that TrianglePacking is contained within the three CVRP variants we consider.

▶ **Definition 4.10** (TrianglePacking)**.** A TrianglePacking instance consists of an undirected graph G such that $|V(G)| = 3q$ vertices, for some integer $q \in \mathbb{N}$. It asks whether there exists a partition $V_1, V_2, \ldots, V_q$ of $V(G)$ so that for each $i \in [q]$, $G[V_i] \simeq K_3$. ⌟

▶ **Theorem 4.11** ([9])**.** TrianglePacking *is* NP*-complete.*

Observe that a partition into triangle graphs is equivalent to a fleet of vehicles that each drive a small tour of just three vertices. We use this intuition in the following reduction.

▶ **Theorem 4.12** (⋆)**.** *All of the following problems are* paraNP*-hard, even with unit demands and weights:*

- LoadCVRP *parameterized by* ℓ*,*
- GasCVRP *parameterized by* g*, and*
- LoadGasCVRP *parameterized by* $\ell + g$*.*

4.3 Hardness from Numerical 3D Matching

In this section we use a reduction from N3DMatching to show that LoadGasCVRP is strongly NP-hard even on trees with constant number of depots, load capacity and unit demands. The reduction can also be adapted to show hardness on trees with constant number of depots, a constant gas constraint and unit edge weights.

▶ **Definition 4.13** (N3DMatching)**.** A *Numerical 3-Dimensional Matching* instance consists of three disjoint sets $X, Y, Z \subseteq \mathbb{N}$, each containing m elements, and a bound $b \in \mathbb{N}$ with $\sum_{a \in X \cup Y \cup Z} a = m \cdot b$. It asks wether $X \cup Y \cup Z$ can be partitioned into m disjoints sets $A_1, \ldots, A_m$, such that each A_i contains exactly one element from each of X, Y, Z and $\sum_{a \in A_i} a = b$. ⌟

▶ **Theorem 4.14** ([9])**.** N3DMatching *is strongly* NP*-complete.*

▶ **Theorem 4.15** (⋆)**.** LoadGasCVRP *is strongly* NP*-hard, even on stars with one depot, constant load capacities and unit demands.*

To show the hardness of LoadGasCVRP on stars with one depot and a constant gas constraint, the construction used in the proof of Theorem 4.15 can be modified. We use client demands instead of edge weights to encode the numbers to be matched and their type in a given N3DMatching instance, changing the load and gas constraints accordingly.

▶ **Corollary 4.16.** LoadGasCVRP *is strongly* NP*-hard, even on stars with one depot and a constant gas constraint.*

4.4 Tractability for Constant Treewidth and Vehicle Capacities

In this section we investigate LoadGasCVRP parameterized by $\mathrm{tw} + \ell + g$. Note that the observation from Lemma 3.7 does not hold once vehicles need to abide capacities, as witnessed in Figure 2.

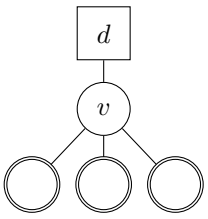


Figure 2 A LoadCVRP instance with $\ell = 1$ where each routing has to use the edge $\{d, v\}$ more than twice.

Thus, the dynamic programming devised to prove Theorem 3.9 cannot directly be applied to LoadGasCVRP. Instead, we track the number of walks with a given set of characteristics separately, leading to an XP running time. Note however, that a similar result to Lemma 3.7 holds for each vehicle individually:

▶ **Lemma 4.17** (⋆)**.** *Let (R, A) be a* LoadGasCVRP *routing, let $p \in R$ and let $H \subseteq G$ be the undirected multigraph representing p. If there is an edge $e \in E(H)$ with multiplicity greater than two, then the walk p' obtained from $H - 2e$ can be used instead of p in R to form a* LoadGasCVRP *routing R' with $w(R') \leq w(R)$.*

Before we formally define the state of the dynamic programming algorithm on the tree decomposition of a given instance for this section, we divert back to BinPacking. We define and solve an extended version of BinPacking, which abstracts from the combination of different sets of walks in order to form new walks fulfilling some desired characteristics. In the algorithm, this operation will be needed for the computation at the Join Nodes of the tree decomposition.

In particular, we encode each walk as a BinPacking item with a *multidimensional size* given by the number of clients and distance traveled. Accordingly, the bins have multidimensional capacities matching the characteristics of the desired walks. As the walks of a partial solution

might have different characteristics, there are several different *bin kinds*. In addition, each walk has a *fingerprint* given by the endpoints of the walk and whether or not it contains a depot. Each bin kind expects a combination of fingerprints fulfilling a certain predicate, according to the endpoints of the desired walk.

▶ **Definition 4.18** (HetMdFpBinPacking)**.** A *Heterogeneous Multidimensional Fingerprint Bin Packing* instance consists of

- a finite set U of items,
- a finite set $\mathcal{F}$ of fingerprints,
- a dimension $d \in \mathbb{N}^+$,
- a number of bin kinds $m \in \mathbb{N}^+$,
- bin capacities $B_1, B_2, \ldots, B_m \in \mathbb{N}^d$ (with $\|B_1\|_1 \geq \|B_2\|_1 \geq \cdots \geq \|B_m\|_1$),
- available bin counts $k_1, k_2, \ldots, k_m$,
- predicates deciding the validity of a multiset of fingerprints $\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_m \colon \mathbb{N}^{\mathcal{F}} \to \{0, 1\}$,
- item sizes $s \colon U \to \mathbb{N}^d \setminus \{0\}$ with $\|s(u)\|_1 \leq \|B_1\|_1$ for all $u \in U$, and
- item fingerprints $F \colon U \to \mathcal{F}$.

It asks whether there exists a partition $\{U_{i,j}\}_{i \in [m], j \in [k_i]}$ of U such that for all i, j

- the bin capacity is not exceeded: $\sum_{u \in U_{i,j}} s(u) \leq B_i$, and
- the multiset of fingerprints is valid: $\mathcal{V}_i(\llbracket F(u) \mid u \in U_{i,j} \rrbracket) = 1$. ⌟

▶ **Theorem 4.19** ($\star$)**.** *There is an algorithm that, given a* HetMdFpBinPacking *instance*

$$\mathcal{B} = (U, \mathcal{F}, d, m, (B_1, B_2, \ldots, B_m), (k_1, k_2, \ldots, k_m), (\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_m), s, F),$$

computes a feasible partition or concludes that there is none in $f(|\mathcal{F}| + d + m + \|B_1\|_1) \cdot (T(|U|) + |\mathcal{B}|^{\mathcal{O}(1)})$ *steps, where* $T \colon \mathbb{N} \to \mathbb{N}$ *is an upper bound on the time complexity of the predicates* $\{\mathcal{V}_i\}_{i \in [m]}$*, for some computable function* f*.*

Note that removing all the additions of HetMdFpBinPacking, the above algorithm can also be used for regular BinPacking.

▶ **Corollary 4.20.** BinPacking *parameterized by the bin capacity* B *is in* FPT*.*

With all preliminaries out of the way, we present the dynamic programming algorithm for LoadGasCVRP. Given a LoadGasCVRP instance $(G, w, D, C, k, \ell, \Lambda, g, r)$ and a nice tree decomposition $\mathcal{T}$ of G, we track partial solutions of the following form:

▶ **Definition 4.21.** Let $t \in \mathcal{T}$. We call a walk p in $G_t^{\downarrow}$ *detached* if $V(p) \cap X_t = \emptyset$ and $V(p) \cap D \neq \emptyset$. We call p *attached* when both endpoints of p are in X_t (but we don't require anything with regards to D). For $u, v \in V(G), \lambda, w \in \mathbb{N}$, an attached u-v-walk with $\Lambda(p) = \lambda$ and $w(p) = w$ is called a (u, v, λ, w)-walk.

Let $S_t = \{s \colon X_t^2 \times [\ell]_0 \times [g]_0 \to \mathbb{N}\}$ be the set of all multisets of combinations of walk endpoints, clients covered, and weight. We refer to the elements $s \in S_t$ as *counters* and abbreviate $\Lambda(s) = \sum_{u,v,\lambda,w} \lambda \cdot s(u, v, \lambda, w)$ and $w(s) = \sum_{u,v,\lambda,w} w \cdot s(u, v, \lambda, w)$.

Let $X \subseteq X_t \cap C$, $c \leq k$, and $s, s^* \in S_t$. We call (X, c, s, s^*) a *solution signature* at t. A set of walks $R = \{p_1, p_2, \ldots, p_{k'}\}$ in $G_t^{\downarrow}$ and assignment of clients $A \colon C \cap (X \cup (V_t^{\downarrow} \setminus X_t)) \to [k']$ is a *partial solution* compatible with (X, c, s, s^*) at t, if

- for all clients $c \in C \cap (X \cup (V_t^{\downarrow} \setminus X_t))$, $c \in V(p_{A(c)})$,
- all walks $p \in R$ are either detached or attached,
- there are exactly c detached walks in R,
- for each $u, v \in X_t, \lambda \in [\ell]_0, w \in [g]_0$,
  - there are exactly $s(u, v, \lambda, w)$ many (u, v, λ, w)-walks $p \in R$ with $V(p) \cap D = \emptyset$, and
  - there are exactly $s^*(u, v, \lambda, w)$ many (u, v, λ, w)-walks $p \in R$ with $V(p) \cap D \neq \emptyset$. ⌟

We use dynamic programming to compute the minimum weight $dp[t, X, c, s, s^*] \in \mathbb{N} \cup \{\infty\}$ of all partial solutions R compatible with (X, c, s, s^*) at all $t \in \mathcal{T}$ and for all solution signatures (X, c, s, s^*) at t. The observant reader might have noticed that since the counters $s \in S_t$ map to the natural numbers, the set S_t is infinite when $X_t \neq \emptyset$. However, we know from Lemma 4.17 that it suffices to look for routings where each of the k walks contain each edge of G at most twice. As there are only finitely many such routings, only finitely many partial solutions leading to these routings can be witnessed at the bags of $\mathcal{T}$. Thus, there are only finitely many counters which are relevant for finding an optimal routing.

First, if the number of covered clients indicated by a counter s is greater than the number of available clients, we know that no compatible partial solution exists. More formally, if

$$\Lambda(s + s^*) > |C \cap (X \cup (V_t^{\downarrow} \setminus X_t))|,$$

then $dp[t, X, \cdot, s, s^*] = \infty$. Note that the above values might still not be equal, as some of the clients in $V_t^{\downarrow}$ might be covered by detached walks. This means that we can limit the computation of dp to only those counters $s \in S_t$ with $s(\cdot, \cdot, \lambda, \cdot) \leq |C|$ for all $\lambda \geq 1$.

This still leaves the value of s unbounded for $\lambda = 0$. However, the dp values for counters with $s(u, v, 0, w) > 1$ can be reduced to the case where $s(u, v, 0, w) = 1$. The fact that a partial solution (R, A) containing a $(u, v, 0, w)$-walk p exists suffices to find the minimum weight partial solution for $s(u, v, 0, w) > 1$. Such a solution can be obtained by repeatedly inserting copies of p into R, which increases the weight of R by $(s(u, v, 0, w) - 1) \cdot w$. Thus, we only need to compute $dp[t, X, c, s, s^*]$ when $s(\cdot, \cdot, 0, \cdot), s^*(\cdot, \cdot, 0, \cdot) \leq 1$.

Before we can start to describe the computation of dp, we need to introduce the concept of *counter reducibility*.

▶ **Definition 4.22.** Let $t \in T$ and $s_1, s_1^*, s_2, s_2^* \in S_t$. The pair of counters (s_1, s_1^*) is reducible to (s_2, s_2^*), denoted as $(s_1, s_1^*) \rightsquigarrow (s_2, s_2^*)$, if for every $X \subseteq X_t$, $c \leq k$, and partial solution (R_1, A_1) compatible with (X, c, s_1, s_1^*) at t, there is a partial solution (R_2, A_2) compatible with (X, c, s_2, s_2^*) at t which can obtained by merging walks of R_1 and updating A_1 accordingly. ⌟

▶ **Lemma 4.23 (⋆).** *Counter reducibility can be decided in $f(|X_t| + \ell + g + z) \cdot n^{\mathcal{O}(1)}$ steps for some function f, where $z = \sum_{u,v \in X_t} (s_2(u, v, 0, 0) + s_2^*(u, v, 0, 0))$.*

The computation of dp then depends on the node type of t:

Leaf Node

If t is a leaf node, then $X_t = \emptyset$, so X, s and s^* must all be empty too. As $V_t^{\downarrow} = \emptyset$, there cannot be any detached walks, so

$$dp[t, X, c, s, s^*] = \begin{cases} 0, & \text{if } c = 0, \\ \infty, & \text{otherwise.} \end{cases}$$

Introduce Vertex Node

Suppose that t is an introduce vertex node with child node t' such that $X_t = X_{t'} \cup \{v\}$. As v is an isolated vertex in $G_t^{\downarrow}$, any attached walk p that contains v must be the empty walk. Thus, for all $u \neq v$ and $w > 0$ we must have $s(u, v, \cdot, \cdot) = s(v, u, \cdot, \cdot) = s^*(u, v, \cdot, \cdot) = s^*(v, u, \cdot, \cdot) = s(v, v, \cdot, w) = s^*(v, v, \cdot, w) = 0$. Depending on whether or not $v \in D$, the empty walks at v are counted in s or s^* respectively. Suppose that $v \notin D$, then $s^*(v, v, \cdot, \cdot) = 0$. One of the empty walks at v should cover a client if and only if v is a client and contained

in X. More formally, for all $\lambda \geq 1$ we have $s(v, v, \lambda, 0) = \mathbb{1}(\lambda = \Lambda(v) \wedge v \in X)$. Lastly, the value $s(v, v, 0, 0)$ can take any value. If all of the above constraints are satisfied, we set $dp[t, X, c, s, s^*] = dp[t', X \setminus \{v\}, c, s_{\downarrow X_{t'}}, s^*_{\downarrow X_{t'}}]$. Otherwise there is no compatible partial solution, and the dp entry is set to ∞. The case $v \in D$ follows analogously with the requirements for s and s^* swapped.

Introduce Edge Node

Suppose that t is an introduce edge node labeled with the edge $e = \{u, v\}$ with a child t'. As seen in Lemma 4.17, any walk in a minimum weight partial solution compatible with (X, c, s, s^*) at t traverses e at most twice. Thus, when removing e, a walk p gets split into at most three subwalks. Let $s', s'^* \in S_{t'}$ and $c_e \in \mathbb{N}$ such that $\max\{c_e, |s'| + |s'^*|\} \leq 3 \cdot (|s| + |s^*|)$. Then a partial solution compatible with (X, c, s, s^*) at t can be constructed from a partial solution compatible with (X, c, s', s'^*) at t' by inserting c_e copies of e if $(s' + c_e \cdot [\![(u, v, 0, w(u, v))]\!], s'^*) \rightsquigarrow (s, s^*)$. Thus, we set

$$dp[t, X, c, s, s^*] = \min_{s', s'^*, c_e} dp[t', X, c, s', s'^*] + c_e \cdot w(u, v),$$

where s', s'^*, c_e range over the values described above.

Forget Node

Suppose that t is a forget node with child node t' such that $X_t = X_{t'} \setminus \{v\}$. A partial solution compatible with (X, c, s, s^*) can interact with v in several ways.

First, there can be detached walks containing v, which are included in the count c. For a given count $c' \leq c$ of detached walks that are also detached from t', let $s_v \in S_{t'}$ be a counter which has $s_v(a, b, \cdot, \cdot) = 0$ whenever $a \neq v$ or $b \neq v$ and $\sum s_v(v, v, \cdot, \cdot) = c - c'$. There are at most $(c - c' + 1)^{(\ell+1)\cdot(g+1)}$ such counters s_v. As the detached walks need to contain a depot, we consider the counter s_v as an addition to s^*.

Second, the walks counted in s and s^* can contain v. However, any such walk cannot start or end at v, as it would otherwise not be attached to X_t. Thus, the counters for t' remain unchanged by this interaction.

In both cases, if $v \in C$, it must be covered regardless of X. Therefore, we set

$$dp[t, X, c, s, s^*] = \min_{c', s_v} dp[t', X \cup (C \cap \{v\}), c', s, s^* + s_v]$$

where c and s_v range over the counts and counters described above.

Join Node

Suppose that t is a join node with children t_1 and t_2. A partial solution compatible with (X, c, s, s^*) is composed of partial solutions from both t_1 and t_2 to cover the clients in $V^{\downarrow}_{t_1}$ and $V^{\downarrow}_{t_2}$ respectively. A walk from a partial solution at t might switch between $V^{\downarrow}_{t_1}$ and $V^{\downarrow}_{t_2}$ a number of times, so it is split into multiple walks at the children. Suppose that a walk p in $G^{\downarrow}_t$ is split into the walks $p_1, p_2, \ldots, p_q$. First, the clients covered by p are distributed among the p_i. There are at most ℓ such client-connecting walks. All walks p_i that do not cover a client are only needed to connect the start and end of two client-connecting walks. As there are $|X_t|$ vertices that could be reached, at most $|X_t|$ walks are needed in between two client-connecting walks. In total, there are thus at most ℓ client-connecting walks and $(\ell + 1) \cdot |X_t|$ walks connecting the endpoints of those. Therefore, we limit the counters

$s_1, s_1^* \in S_{t_1}, s_2, s_2^* \in S_{t_2}$ to those which are bounded above by $(\ell+1)\cdot|X_t|\cdot(|s|+|s^*|)$. Similarly to before, only the counters which satisfy $(s_1+s_2, s_1^*+s_2^*) \rightsquigarrow (s, s^*)$ are considered. Note that the clients in X are split between the two partial solutions so that the partial solution in t_1 covers some subset $X_1 \subseteq X$ and leaves $X \setminus X_1$ to t_2's solution. Similarly, the number of detached walks is split into $c_1 \leq c$ and $c - c_1$. Bringing it all together, we set

$$dp[t, X, c, s, s^*] = \min_{\substack{X_1 \subseteq X, c_1 \leq c, \\ s_1, s_1^*, s_2, s_2^*}} dp[t_1, X_1, c_1, s_1, s_1^*] + dp[t_2, X \setminus X_1, c - c_1, s_2, s_2^*],$$

where s_1, s_1^*, s_2, s_2^* range over the values described above.

▶ **Theorem 4.24** (⋆). LoadGasCVRP *parameterized by* $\mathrm{tw} + \ell + g$ *is in* XP.

Furthermore, the algorithm from Theorem 4.24 can be modified to yield similar results for LoadCVRP and GasCVRP. In particular, the counters S_t for each node $t \in \mathcal{T}$ should only track the endpoints of each walk and one of the two constraints.

▶ **Corollary 4.25.** LoadCVRP *parameterized by* $\mathrm{tw} + \ell$ *and* GasCVRP *parameterized by* $\mathrm{tw} + g$ *are in* XP.

5 Conclusion

We have presented detailed tractability results for vehicle routing problems regarding the parameters given in the input and the treewidth of the underlying network. We have shown complete results for all mentioned parameters in the uncapacitated case. For VRP treewidth as a parameter suffices for the existence of FPT algorithms. On the other hand, once any form of capacity constraints is introduced, vehicle routing becomes hard even on trees. Combining treewidth with other parameters is only sensible for parameters which are not sufficient for FPT algorithms on their own, namely the capacity, k and $|D|$. Out of these parameters, only the inclusion of all present capacity parameters yields tractability.

While we also proved complexity bounds of the capacitated vehicle routing variants, some questions remain unanswered. We have provided an XP algorithm for treewidth and capacity as a combined parameter. However, the existence of an FPT algorithm for the problem could neither be ruled out nor proven in this work. Additionally, while the W[1]-hardness for the parameter $\mathrm{tw} + k + |D|$ is likely to rule out an FPT-algorithm, future work could explore XP algorithms for this parameter.

References

1 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.

2 Matthias Bentert, Fedor V. Fomin, Petr A. Golovach, Tuukka Korhonen, William Lochet, Fahad Panolan, M. S. Ramanujan, Saket Saurabh, and Kirill Simonov. Packing short 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 1425–1463. SIAM, 2025. doi:10.1137/1.9781611978322.45.

3 Hans L Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Information and Computation*, 243:86–111, 2015. doi:10.1016/J.IC.2014.12.008.

4 Kris Braekers, Katrien Ramaekers, and Inneke Van Nieuwenhuyse. The vehicle routing problem: State of the art classification and review. *Computers & industrial engineering*, 99:300–313, 2016. doi:10.1016/J.CIE.2015.12.007.

5 Marek Cygan, Fedor V Fomin, Lukasz 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.

6 G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management Science*, 6:80–91, October 1959.

7 Mark de Berg, Hans L. Bodlaender, Sándor Kisfaludi-Bak, and Sudeshna Kolay. An eth-tight exact algorithm for euclidean tsp. *SIAM Journal on Computing*, 52(3):740–760, 2023. doi:10.1137/22M1469122.

8 Michelle Döring, Jan Fehse, Tobias Friedrich, Paula Marten, Niklas Mohrin, Kirill Simonov, Farehe Soheil, Jakob Timm, and Shaily Verma. Parameterized complexity of vehicle routing. *arXiv preprint arXiv:2509.10361*, 2025.

9 Michael R Garey and David S Johnson. *Computers and intractability*, volume 174. freeman San Francisco, 1979.

10 Klaus Jansen, Stefan Kratsch, Dániel Marx, and Ildikó Schlotter. Bin packing with fixed number of bins revisited. *Journal of Computer and System Sciences*, 79(1):39–49, 2013. doi:10.1016/j.jcss.2012.04.004.

11 Andrea Mor and Maria Grazia Speranza. Vehicle routing problems over time: a survey. *Annals of Operations Research*, 314(1):255–275, 2022. doi:10.1007/S10479-021-04488-0.

12 Ted K Ralphs, Leonid Kopman, William R Pulleyblank, and Leslie E Trotter. On the capacitated vehicle routing problem. *Mathematical programming*, 94:343–359, 2003. doi:10.1007/S10107-002-0323-0.

13 Neil Robertson and Paul D Seymour. Graph minors. III. Planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49–64, 1984. doi:10.1016/0095-8956(84)90013-3.

14 Šimon Schierreich and Ondřej Suchý. Waypoint routing on bounded treewidth graphs. *Information Processing Letters*, 173:106165, 2022. doi:10.1016/j.ipl.2021.106165.

Parameterized Algorithms for Diversity of Networks with Ecological Dependencies

Mark Jones
TU Delft, The Netherlands

Jannik Schestag
TU Delft, The Netherlands

Abstract

For a phylogenetic tree, the phylogenetic diversity of a set A of taxa is the total weight of edges on paths to A. Finding small sets of maximal diversity is crucial for conservation planning, as it indicates where limited resources can be invested most efficiently. In recent years, efficient algorithms have been developed to find sets of taxa that maximize phylogenetic diversity either in a phylogenetic network or in a phylogenetic tree subject to ecological constraints, such as a food web. However, these aspects have mostly been studied independently. Since both factors are biologically important, it seems natural to consider them together.

In this paper, we introduce decision problems where, given a phylogenetic network, a food web, and integers k, and D, the task is to find a set of k taxa with phylogenetic diversity of at least D under the *maximize all paths* measure, while also satisfying viability conditions within the food web. Here, we consider different definitions of viability, which all demand that a "sufficient" number of prey species survive to support surviving predators.

We investigate the parameterized complexity of these problems and present several fixed-parameter tractable (FPT) algorithms. Specifically, we provide a complete complexity dichotomy characterizing which combinations of parameters – out of the size constraint k, the acceptable diversity loss $\overline{D}$, the scanwidth of the food web $\mathrm{sw}_{\mathcal{F}}$, the maximum in-degree δ in the network, and the network height h – lead to W[1]-hardness and which admit FPT algorithms.

Our primary methodological contribution is a novel algorithmic framework for solving phylogenetic diversity problems in networks where dependencies (such as those from a food web) impose an order, using a color coding approach.

2012 ACM Subject Classification Applied computing → Computational biology; Theory of computation → Fixed parameter tractability

Keywords and phrases Phylogenetic Diversity, Fixed-Parameter Tractability, Phylogenetic Networks, Food Webs, Color Coding

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.11

Related Version *Full Version*: `http://arxiv.org/abs/2510.09512` [18]

Funding *Mark Jones*: Partially funded by the Dutch Organisation for Scientific Research (NWO) grant OCENW.KLEIN.125 and OCENW.M.21.306.
Jannik Schestag: Funded by the Dutch Research Council (NWO), project "Optimization for and with Machine Learning (OPTIMAL)" OCENW.GROOT.2019.015.

1 Introduction

The Sixth Mass Extinction eliminates species and their genera at an unprecedented rate [5], even exceeding rates in previous mass extinction events [27]. The situation is severe enough that approximately a quarter of Earth's existing species are at threat [10]. Inaction now means jeopardizing further parts of the animal tree of life.

© Mark Jones and Jannik Schestag;
licensed under Creative Commons License CC-BY 4.0

Because resources are not sufficiently available to preserve all species, scientists developed the *phylogenetic diversity* measure [8] to cover the necessity of making an educated decision on which set of species the limited resources should be invested in. Given a phylogenetic tree with edge weights in which leaves represent present-day species, the phylogenetic diversity of a set of species A is the total weight of edges on paths from the root to species in A. Intuitively, a set of species with larger phylogenetic diversity captures a larger variety of genetic material, and is therefore expected to have larger biodiversity. Phylogenetic diversity became the most popular measure of the significance of a set of species [33].

Apart from its biological relevance, phylogenetic diversity probably also found favor in the eyes of many scientists for its highly desirable trait of being easy to compute [8]. Indeed, an optimal solution for maximizing phylogenetic diversity can be found with a greedy algorithm [25, 30] and therefore even large instances can be solved within seconds [22].

To model further relevant aspects of conservation planning, more problems were defined, in which a special focus was placed upon capturing varying costs of saving species [20, 26], finding optimal conservation areas [3, 23], considering species' extinction times [17], or preserving viable sets of species [23]. In the latter problem a food web, modeling predator-prey relationships, is given in addition to the phylogenetic tree. It is asked to find a set A of species that maximizes the phylogenetic diversity and is viable in the sense that each species in A is either a source in the ecosystem, or finds prey within A. Due to concerns that one prey could be insufficient, further definitions of viability have been defined [28].

The notion of phylogenetic trees has been generalized. Ancestry of species is in recent years more often modeled with phylogenetic networks, which in contrast to phylogenetic trees also allow hybridization events or horizontal gene transfer [15]. Consequently, generalizations of phylogenetic diversity on networks have been proposed [4, 16, 31, 32, 34].

As phylogenetic networks represent the connection between species a lot better than phylogenetic trees and considering viability constraints are vital, it is natural to combine these two aspects. Yet, to the best of our knowledge, this has not been done so far. Therefore, we take the step and define problems for the maximization of phylogenetic diversity on networks with different viability definitions. As it was expected that "a combination of these concepts [would] result in very hard problems." [28], we turn to the toolbox of parameterized complexity to break intractability.

Parameterized complexity is one method to cope with NP-hardness. In this, we consider a problem Π and a parameter p of size κ and say that (Π, p) is FPT if Π can be solved in $\mathcal{O}^*(f(\kappa))$ time, for some computable function f. If Π is W[1]-hard with respect to p or even NP-hard if κ is a constant, then the existence of an FPT-algorithm is unlikely [6, 7].

We consider three definitions of viability, based on whether a non-source species in the food web needs to have one or all of its prey available, or whether a weighted sum of the available prey must reach a certain threshold. We provide a complete complexity dichotomy, for the latter two of these definitions, in the sense that for every combination of the following parameters, we show whether the defined problems are W[1]-hard, or can be solved with an FPT-result: the size constraint k; the acceptable diversity loss $\overline{D}$; the scanwidth of the food web $\mathrm{sw}_{\mathcal{F}}$; the maximum in-degree δ in the network; and the network height h. For the other definition of viability, we have a near-complete complexity dichotomy that omits only one of the above combinations. In particular, we provide FPT algorithms for the most general version of the problem we define for the parameters $\overline{D} + \mathrm{sw}_{\mathcal{F}}$ and $k + \mathrm{sw}_{\mathcal{F}} + \delta + h$. For the former, we depend on a notion of *anchors* already used in [17], but we improve on their algorithmic idea, in two ways. First, we use a color coding technique which only requires one color per edge. Secondly, we consider a non-linear order. By this, we are able to even provide algorithms for the smaller parameter $\overline{k} + \mathrm{sw}_{\mathcal{F}}$ for the version of the problem on trees.

2 Preliminaries

2.1 Definitions

We use the $\mathcal{O}^*$-notation which omits factors polynomial in the input size. For a positive integer a, by $[a]$ we denote the set $\{1, 2, \ldots, a\}$, and by $[a]_0$ the set $\{0\} \cup [a]$. For functions $f : A \to B$, where B is a family of sets, we define $f(A') := \bigcup_{a \in A'} f(a)$, and if $B \subseteq \mathbb{Q}$, we define $f_\Sigma(A') := \sum_{a \in A'} f(a)$, for subsets $A' \subseteq A$. We write $\{u, v\}$ for an undirected edge between u and v and uv for a directed edge from u to v. For any graph G, we write $V(G)$ and $E(G)$ for the set of vertices or edges, respectively. For a set of edges E', we write $V(E')$ for the vertices with at least one endpoint in an edge of E'. For a vertex set $V' \subseteq V(G)$, we let $G[V'] := (V', \{e \in E(G) \mid V(\{e\}) \subseteq V'\})$ denote the subgraph of G induced by V'. We generalize to edge sets $E' \subseteq E(G)$ by $G[E'] := G[V(E')]$.

Phylogenetic Networks and Phylogenetic Diversity. For a given set X, a *phylogenetic X-network* $\mathcal{N} = (V, E, \omega)$ is a directed, acyclic graph (V, E) with *edge-weight* $\omega : E \to \mathbb{N}_{>0}$, in which there is a single vertex with an in-degree of 0, the *root* ρ, and X is the set of vertices with an in-degree of 1 and an out-degree of 0, called the *leaves*. All other vertices split up into *tree vertices*, which have an in-degree of 1 and an out-degree of at least 2, and *reticulations*, which have an out-degree of 1 and an in-degree of at least 2. Edges incoming at reticulations (tree vertices) are *reticulation edges* (*tree edges*). The set of reticulations, tree vertices, reticulation and tree edges are denoted with $V_R(\mathcal{N})$, $V_T(\mathcal{N})$, $E_R(\mathcal{N})$, and $E_T(\mathcal{N})$, respectively. A *phylogenetic X-tree* $\mathcal{T} = (V, E, \omega)$ is a phylogenetic X-network without reticulations. The set X is a set of *taxa* (species). We interchangeably use the words taxon and leaf. In biological applications, the set X is a set of taxa, and the other vertices of $\mathcal{N}$ correspond to biological ancestors of these taxa. An edge $e = uv$ represents direct biological inheritance from u to v. The weight $\omega(e)$ describes the phylogenetic distance between the endpoints of e. As these endpoints correspond to distinct, (possibly extinct) species, we may assume this distance is greater than zero. Reticulations correspond to species that have direct inheritance from multiple ancestors, such as hybrid species.

For a vertex $v \in V$, the *offspring* $\mathrm{off}(v)$ of v is the set of leaves $x \in X$ for which there is a path from v to x. Given a set of taxa $A \subseteq X$, let $E(A)$ denote the set of edges $uv \in E$ with $\mathrm{off}(v) \cap A \neq \emptyset$. The *phylogenetic diversity* $PD_\mathcal{N}(A)$ of A is defined by

$$PD_\mathcal{N}(A) := \sum_{e \in E(A)} \omega(e). \tag{1}$$

In other words, the phylogenetic diversity $PD_\mathcal{N}(A)$ of a set A of taxa is the sum of the weights of edges that are on a path to a taxon in A.

For phylogenetic trees, this measure is well established [8]. For phylogenetic networks the search for the most relevant measure is still ongoing, but so far the above definition, also called *All-Paths-PD*, is the measure that is "the simplest" [4, 16, 31, 32, 34] and is the only measure of phylogenetic diversity considered in this paper.

Food Webs. A *food web* $\mathcal{F} = (X, E, \gamma)$ *on* X for a set of taxa X, is a directed acyclic graph (X, E) with an edge weight function $\gamma : E \to [0, 1]$.

For an edge $xy \in E$, we say that x is *prey* of y and y is a *predator* of x. Thus, edges in $\mathcal{F}$ are directed from prey to predator. The set of prey and predators of x is denoted with $\mathrm{prey}(x)$ and $\mathrm{pred}(x)$, respectively. The set $\mathrm{prey}^{(E)}(x)$ is the set of edges incoming at v and $\mathrm{pred}^{(E)}(x)$ is the set of edges outgoing of v in $\mathcal{F}$. Taxa without prey are *sources*. For the

problems considered in this paper, instances in which the food web has several sources can in quadratic time be transformed into one where there is only one source [21, Observation 2.3]. Therefore, throughout the paper, we assume that $\mathcal{F}$ only has a single source $s_{\mathcal{F}}$.

For a given food web $\mathcal{F}$, a set of taxa $A \subseteq X$ is *γ-viable* if $\gamma_\Sigma(\{ux \mid u \in \text{prey}(x) \cap A\}) \geq 1$ for each non-source $x \in A$. That is, the total weight of edges incoming from taxa in A is at least 1 [28]. Given also a set $E' \subseteq E(\mathcal{F})$ of edges, a set of taxa $A \subseteq X$ is *E'-part-γ-viable* if $\gamma_\Sigma(\{ux \in \text{prey}^{(E)}(x) \mid u \in A \text{ or } ux \in E'\}) \geq 1$ for each non-source $x \in A$. That is, the total weight of edges incoming from taxa in A together with incoming edges in E' is at least 1.

We consider two important special cases of viability. If $\gamma(e) = 1$ for all $e \in E$, then we say a γ-viable set $A \subseteq X$ is *ε-viable*. That is equivalent to saying that A is ε-viable, if each non-source in A can find prey in A. If $\gamma(ux) = 1/|\text{prey}(x)|$ for every edge in $\text{prey}^{(E)}(x)$ for each $x \in X$, then we say a γ-viable set $A \subseteq X$ is *1-viable*. That is, A is 1-viable, if all prey of each taxon in A are in A.

Problem Definitions. In the classical Maximize Phylogenetic Diversity (Max-PD) problem, we are given a phylogenetic tree $\mathcal{T}$, and integers k, and D, and it is asked whether a set of k taxa with a phylogenetic diversity of at least D exists [8].

The problems Map-PD [4], ε-PDD [23], and 1-PDD [28] are generalizations of Max-PD in the following sense. In Map-PD, we are given a phylogenetic network instead of a tree and the question stays the same – just with the more general phylogenetic diversity measure. In ε-PDD, 1-PDD and Weighted-PDD, we are, in addition to the input of Max-PD, given a food web and the solution set is required to be ε-viable, 1-viable or γ-viable, respectively.

In this paper, we consider the following generalizations of these problems.

Map-Weighted-PDD
Input: A phylogenetic X-network $\mathcal{N}$, a food web $\mathcal{F}$ on X, and integers $k, D \in \mathbb{N}$.
Question: Is there an γ-viable set $S \subseteq X$ such that $|S| \leq k$, and $PD_{\mathcal{N}}(S) \geq D$?

We call the set S a *solution* of the instance. The problems Map-ε-PDD and Map-1-PDD are defined analogously, where the set S is required to be ε-viable and 1-viable, respectively.

We note that in ε-PDD, 1-PDD, and Weighted-PDD, as originally defined, the phylogenetic network is required to be a tree.

Throughout the paper, we adopt the convention that n is the number of taxa $|X| = |V(\mathcal{F})|$ and that m is the number of edges of the food web $|E(\mathcal{F})|$.

Scanwidth. For a directed, acyclic graph $G = (V, E)$, a rooted, directed tree $T = (V, E')$ is a *tree extension of G* if for each edge $uv \in E$ there is a path from u to v in T. We say that an edge $uv \in E$ *passes over* an edge $e \in E'$ if the (only) path from u to v in T contains e. For an edge $uv \in E'$, the set of edges that pass over uv is $GW(v)$ and $T_{\mathcal{F}}^{(v)}$ is the set of vertices that can be reached from v in $T_{\mathcal{F}}$. The *scanwidth of a tree extension* T of G is the maximum number of edges of G that pass over an edge of T. The *scanwidth of G* is the minimal scanwidth of any tree extension of G [2]. Computing the tree extension and scanwidth of a directed, acyclic graph is NP-hard [2] and, when considered for phylogenetic networks, FPT when parameterized by the level of the network [11, 13]. In this paper, we will use the parameter *scanwidth* $\text{sw}_{\mathcal{F}}$ *of the food web*, and assume we are given a tree extension of $\mathcal{F}$ of minimum scanwidth.

Other Main Parameters. Here, we define the main parameters which are used in this paper.

For an instance $\mathcal{I} = (\mathcal{N}, \mathcal{F}, k, D)$ of Map-Weighted-PDD, we define $\overline{k} := |X| - k$ and $\overline{D} := PD_{\mathcal{N}}(X) - D = \sum_{e \in E} \omega(e) - D$. Observe that if a set S of k taxa with diversity D is preserved, then $\overline{k}$ taxa are not in S and a diversity of $\overline{D}$ is lost. We therefore speak of $\overline{D}$ as the *acceptable loss of diversity*.

We define δ to be the *maximum in-degree of a reticulation* of $\mathcal{N}$. We define h_R and h_T to be the maximum number of reticulations and tree vertices, respectively, which are in a path in $\mathcal{N}$. Further, we define $h := h_R + h_T$ to be the *height of the network*.

Color Coding. In this paper, we use color coding methods. For an in-depth treatment of color coding, we refer the reader to [6, Chapter 5] and [1].

▶ **Definition 2.1.** *For integers n and k, an (n,k)-*perfect hash family *$\mathcal{H}$ is a family of mappings $f : [n] \to [k]$ such that for every subset Z of $[n]$ of size at most k, there is an $f \in \mathcal{H}$ which is injective when restricted to Z.*

Each mapping in an (n,k)-perfect hash family can be seen as coloring of a set of n items into k colors. If we are interested in finding a set of at most k items satisfying some property, then we may assume that each element in the set will have a different color under some coloring. This assumption can make solving a problem easier, and we will use it in the algorithms developed in Sections 3 and 4.

▶ **Proposition 2.2** ([6, 24])**.** *For any integers $n, k \geq 1$, a (n,k)-perfect hash family of size $e^k k^{\mathcal{O}(\log k)} \cdot \log n$ can be constructed in time $e^k k^{\mathcal{O}(\log k)} \cdot n \log n$.*

2.2 Related work

ε-PDD, defined in [23], was conjectured to be NP-hard [29]. A formal prove appeared in [9].

▶ **Theorem 2.3** ([9, Theorem 5.1])**.** *ε-PDD is NP-hard even if the phylogenetic tree has a height of 2 and the food web is an out-tree.*

ε-PDD has been analyzed within parameterized complexity [21] and it has been shown that ε-PDD is FPT when parameterized with D, but W[1]-hard with respect to $\overline{D}$ [21]. In [28], further definitions for viability were given, among them γ-viable and 1-viable. It has been shown that 1-PDD is W[1]-hard with respect to k, D, $\overline{k}$, and $\overline{D}$ [12]. Further, ε-PDD [21] and Weighted-PDD [28] were analyzed with respect to parameters categorizing the structure of the food web.

▶ **Theorem 2.4.**

(a) *ε-PDD is W[1]-hard when parameterized with $\overline{k}$ or $\overline{D}$, even if the phylogenetic tree is a star [21, Proposition 5.1].*

(b) *1-PDD is W[1]-hard when parameterized with $\overline{k}$ or $\overline{D}$, even if the phylogenetic tree is a star [12, Theorem 3.3].*

(c) *1-PDD is W[1]-hard when parameterized with k or D, even if the phylogenetic tree is a star [12, Theorem 3.2].*

In recent years, the question of how to model phylogenetic diversity in networks best has drawn some attention [4, 31, 32, 34]. The measure *All-Paths-PD*, as defined in [4], is hereby the easiest to understand and also computationally slightly less challenging than other definitions. Map-PD is FPT when parameterized with D, and can be solved in $\mathcal{O}^*(2^{\text{ret}_{\mathcal{N}}})$ time for $\text{ret}_{\mathcal{N}}$ being the number of reticulations, in $\mathcal{O}^*(2^{\text{tw}_{\mathcal{N}}})$ for $\text{tw}_{\mathcal{N}}$ the treewidth of the network [16], and in $\mathcal{O}^*(2^{\text{sw}_{\mathcal{N}}})$ time for $\text{sw}_{\mathcal{N}}$ the scanwidth of the network [14].

Table 1 This table shows parameterized complexity results. Except for the marked cell, Map-1-PDD and Map-ε-PDD have the same complexity result. All FPT results are new. For any set of taxa A, in polynomial time $PD_{\mathcal{N}}(A)$ and whether A is γ-viable can be computed. Consequently, by iterating over all subsets of X, Map-Weighted-PDD is trivially FPT for $n = k + \overline{k} < k + \overline{D}$. Entry *: Map-1-PDD is W[1]-hard when parameterized with $k+\delta+h$ or even $D+\delta+h$ (Thm. 2.4c) while we conjecture Map-ε-PDD to be FPT when parameterized with $k + \delta + h$ (Con. 4.9).

$\mathrm{sw}_{\mathcal{F}}$	δ	h	parameter alone		k + parameter		$\overline{D}$ + parameter	
✗	✗	✗	*no parameter*		W[2]-h	Thm. 2.5,[4, 16]	W[1]-h	Thm. 2.4a&b,[21, 12]
✗	✗	✓	p-NP-h	Thm. 2.3,[9]	W[2]-h	Thm. 2.5,[4, 16]	W[1]-h	Thm. 2.4a&b,[21, 12]
✗	✓	✗	p-NP-h	Thm. 2.3,[9]	W[2]-h	Cor. 2.6	W[1]-h	Thm. 2.4a&b,[21, 12]
✗	✓	✓	p-NP-h	Thm. 2.3,[9]	* *see caption*		W[1]-h	Thm. 2.4a&b,[21, 12]
✓	✗	✗	p-NP-h	Thm. 2.3,[9]	W[2]-h	Thm. 2.5,[4, 16]	FPT	Cor. 3.2a
✓	✗	✓	p-NP-h	Thm. 2.3,[9]	W[2]-h	Thm. 2.5,[4, 16]	FPT	Cor. 3.2a
✓	✓	✗	p-NP-h	Thm. 2.3,[9]	W[2]-h	Cor. 2.6	FPT	Cor. 3.2a
✓	✓	✓	p-NP-h	Thm. 2.3,[9]	FPT	Cor. 4.3	FPT	Cor. 3.2a

▶ **Theorem 2.5** ([4, 16]). Map-PD *is* W[2]*-hard, when parameterized with the solution size k, even if every path from the root to a leaf contains exactly one tree vertex and one reticulation.*

Recently, Map-PD has been considered in semidirected networks. In semi directed networks, Map-PD can be solved in $\mathcal{O}^*(2^\ell)$ time, where ℓ is the level of the network [14].

Our Contribution. We analyze Map-ε-PDD and Map-1-PDD with respect to the parameters k, $\overline{D}$, $\mathrm{sw}_{\mathcal{F}}$, δ, and h and show, for any combination of these parameters, whether Map-1-PDD is W[1]-hard, or admits an FPT algorithm. For Map-ε-PDD, we only leave one case as a conjecture and prove all others. All algorithms we prove for the generalization Map-Weighted-PDD. Parameterized results for Map-ε-PDD and Map-1-PDD are displayed in Table 1.

In Section 3, we prove that Map-Weighted-PDD is FPT with respect to $\overline{D} + \mathrm{sw}_{\mathcal{F}}$. In Section 4, we show that Map-Weighted-PDD is FPT with respect to $k + \mathrm{sw}_{\mathcal{F}} + h + \delta$. If any of these parameters is dropped, Map-1-PDD is W[1]-hard.

The algorithm presented in Section 3 is our primary methodological contribution. In this novel approach, only a single color per edge is used in a color coding algorithm. This can be used to prove that ε-PDD and 1-PDD are FPT with respect to $\overline{k} + \mathrm{sw}_{\mathcal{F}}$, see Corollary 3.2b. We expect this to be applicable for similar problems parameterized with $\overline{k}$ as well.

Due to space restrictions, proofs of theorems and lemmas marked with $\star$ are partly or fully deferred to the appendix.

2.3 Preliminary Observations

By Theorem 2.5 Map-PD is W[2]-hard with respect to k even for a network of a small height [16]. In this hardness reduction, however, the maximal in-degree of reticulations is big. It is an easy observation, that, in polynomial time, one can replace vertices of a large degree with a stack of vertices, where the newly created edges have negligible weight compared to the original edges of the network. This works for reticulations and tree vertices alike. We obtain the following result.

▶ **Corollary 2.6.** Map-PD *is* W[2]*-hard with respect to k even if the network is binary.*

If the food web is an out-tree, each taxon $x \neq s_{\mathcal{F}}$ has exactly one prey. We conclude that the result of Theorem 2.3 also holds for 1-PDD.

▶ **Corollary 2.7.** 1-PDD *is* NP*-hard even if the phylogenetic tree has a height of 2 and the food web is an out-tree.*

3 Parameter $\overline{D} + \mathrm{sw}_{\mathcal{F}}$

By Theorem 2.4a and b, ε-PDD and 1-PDD are W[1]-hard when parameterized by $\overline{D}$. In this section, we prove that Map-ε-PDD and Map-1-PDD are FPT when parameterized by $\overline{D} + \mathrm{sw}_{\mathcal{F}}$. Further, we show that when the phylogenetic network is a tree, even an FPT-running time with respect to the smaller parameter $\overline{k} + \mathrm{sw}_{\mathcal{F}}$ is possible. To prove these results, we present our main methodological contribution. This approach uses *anchors*, as introduced in [17]. Yet, it is new to use this approach either on phylogenetic networks or for the smaller parameter $\overline{k}$.

▶ **Theorem 3.1** ($\star$)**.** *Let a tree extension $T_{\mathcal{F}}$ of the food web with scanwidth $\mathrm{sw}_{\mathcal{F}}$ be given.*

(a) Map-Weighted-PDD *can be solved in $\mathcal{O}(2^{7.530\cdot\overline{D}+\mathrm{sw}_{\mathcal{F}}+\mathcal{O}(\log^2(\overline{D}))}\cdot n\cdot|E(\mathcal{N})|\cdot\log|E(\mathcal{N})|)$ time.*

(b) Weighted-PDD *can be solved in $\mathcal{O}(2^{15.059\cdot\overline{k}+\mathrm{sw}_{\mathcal{F}}+\mathcal{O}(\log^2(\overline{k}))}\cdot n\cdot|E(\mathcal{N})|\cdot\log|E(\mathcal{N})|)$ time.*

As a consequence of this theorem, we obtain these results.

▶ **Corollary 3.2.**

(a) Map-ε-PDD *and* Map-1-PDD *are* FPT *with respect to $\overline{D} + \mathrm{sw}_{\mathcal{F}}$.*

(b) ε-PDD *and* 1-PDD *are* FPT *with respect to $\overline{k} + \mathrm{sw}_{\mathcal{F}}$.*

For the remainder of this section, fix a tree extension $T_{\mathcal{F}}$ of the food web. Let $x_1, \dots x_n$ be an ordering of the taxa such that if x_i is a parent of x_j in $T_{\mathcal{F}}$ then $i < j$, and if x_i and x_j share a parent in $T_{\mathcal{F}}$ with $i < j$, then every vertex in the subtree of $T_{\mathcal{F}}$ rooted x_i appears before every vertex in the subtree of $T_{\mathcal{F}}$ rooted x_j in the ordering. Such an ordering can be found by taking a depth-first traversal of $T_{\mathcal{F}}$. For a set of edges F in a directed acyclic graph G, we say an edge $e = uv \in F$ is a *highest edge in F* if no incoming edge of u is in F, and similarly we say $uv \in F$ is a *lowest edge in F* if no outgoing edge of v is in F. For a vertex u with outgoing edges uv and uv', we say uv' is a *sibling edge of uv*.

The main ideas of our proof is as follows. Our approach uses color coding techniques, wherein we generate a number of colorings on the edges of $\mathcal{N}$, and seek a solution under the assumption that a certain set fulfills that each edge in the set has a different color.

Our dynamic programming algorithm will search for a structure that we define below as *perfect triples*. Roughly speaking, a perfect triple (A, χ_1, χ_2) consists of a set of taxa A such that $X \setminus A$ is a solution for our instance of Map-Weighted-PDD, and colorings χ_1, χ_2 assigning each leaf in A to a set of colors. Suppose the leaves of A are 'killed' one at a time, in order determined by the depth-first traversal of $T_{\mathcal{F}}$. Then as each leaf is deleted, a certain set of edges will be 'lost' in the sense that they no longer have any offspring that have not been killed. For each $x \in A$, the set $\chi_1(x)$ corresponds to the colors of the edges that are lost when x is deleted. Furthermore, for each highest edge e that is lost when x is killed, there must be a corresponding sibling edge e' that has not yet been lost (otherwise the parent edge(s) of e would also be lost. The colors of these sibling edges are represented by the set $\chi_2(x)$. Note that some of these sibling edges may themselves be lost when a later leaf from A is killed. We may assume, via standard color-coding techniques, that all the lost edges and sibling edges together are multicolored.

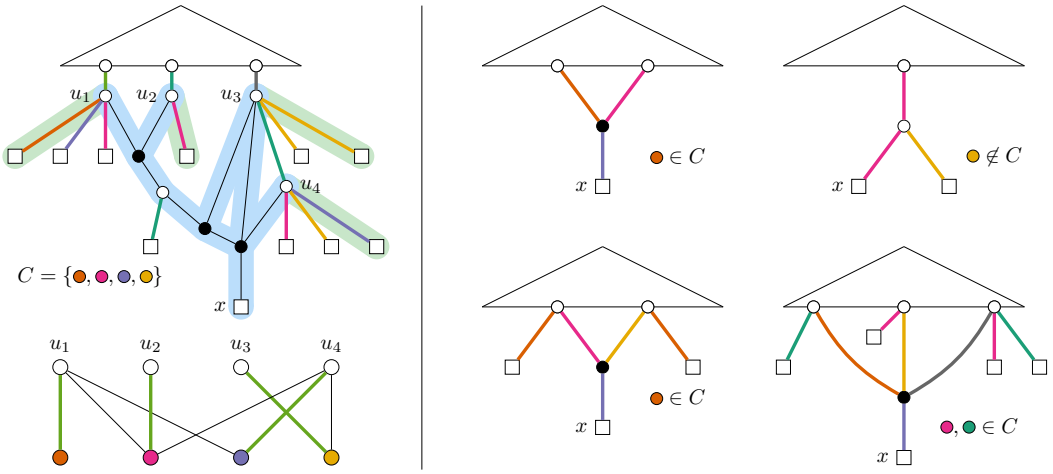


Figure 1 Top Left: An example network. The set $F_{x,C}$ is highlighted in ○ and one possible set $F'_{x,C}$ in ○. For the sake of readability, colors of edges in $F_{x,C}$ are omitted. Bottom Left: The bipartite graph which is constructed in Lemma 3.5. A perfect matching is highlighted.
Right: Four examples of networks, where $F_{x,C}$ is not a respecting set of edges.

Our dynamic programming algorithm will keep track of the existence of perfect triples satisfying certain properties. In particular, we store the minimum possible total weight of the set of lost edges for a perfect triple.

We now give some formal definitions.

▶ **Definition 3.3.** *Fix an integer N and a mapping $c : E_T(\mathcal{N}) \to [N]$. For a taxon $x \in X$ and a set of colors $C \subseteq [N]$, a set of edges $F \subseteq E_T(\mathcal{N})$ is* (x, C)-respecting, *if*

- *F contains the edge incoming at x,*
- *$c(e) \notin C$ for each $e \in F$,*
- *for each $uv \in F$ there is a directed path from v to x within $\mathcal{N}[F]$, and*
- *there is a set F' of edges, which we call* anchors of F, *such that*
  - *$c(e) \in C$ for each edge $e \in F'$,*
  - *$c(e_1) \neq c(e_2)$ for each pair $e_1, e_2 \in F \cup F'$, and*
  - *for each edge $uv \in F$, exactly one of the following is true*
  - **(1)** *F contains all edges incoming at u and $c(e) \notin C$ for each edge e outgoing of u.*
  - **(2)** *F does not contain any edge incoming at u and there is an edge e outgoing of u with $c(e) \in C$ and $e \in F'$.*

When (x, C) is clear from the context we say an (x, C)-respecting set is simply respecting.

In Figure 1 an example of a respecting set of edges is given and some example networks for which x under a certain coloring has no respecting set of edges.

We continue with proving some essential properties about respecting sets.

▶ **Lemma 3.4.** *For each taxon $x \in X$ and each set $C \subseteq [N]$ of colors, at most one set of edges is (x, C)-respecting.*

Proof. Towards a contradiction, assume that F_1 and F_2 are (x, C)-respecting and that there is $u'v' \in F_1$ such that $u'v' \notin F_2$. Fix a set F'_1 of anchors of F, and a set F'_2 of anchors of F_2. Choose any path from v' to x in $\mathcal{N}[F_1]$ and let uv be the first edge on this path that occurs in F_2, also. Such an edge exists as the edge incoming at x is in any respecting set of edges. As F_2 does not contain all edges incoming at u, there is an edge $uw \in F'_2$ with $c(uw) \in C$. But then F_1 is not respecting, as F_1 contains an edge incoming at u. ◀

As a consequence of Lemma 3.4, we write $F_{x,C}$ for the unique set of (x, C)-respecting edges, if existent. If the set $F_{x,C}$ exists, we write $c(F_{x,C})$ for the colors on edges in $F_{x,C}$. By definition, $c(F_{x,C})$ and C are disjoint. If the set $F_{x,C}$ does not exist, we define $\omega(F_{x,C}) = \infty$. We note that the set of anchors is not necessarily unique.

▶ **Lemma 3.5.** *For a taxon $x \in X$ and a set of colors $C \subseteq [N]$, in $\mathcal{O}(|E(\mathcal{N})| \cdot \sqrt{N})$ time, we can compute $F_{x,C}$, or conclude that it does not exist.*

Proof. We construct a unique set of edges F such that either F is respecting for x and C, or such a set does not exist, as follows. Initially let F contain the unique incoming edge e at x. Now, for each edge uv in F in turn, if u has no outgoing edge uv' with $c(uv') \in C$, then add all incoming edges of u to F. Repeat this process exhaustively to complete the construction of F. If $c(e) \notin C$ for some $e \in F$ or if two edges have the same color, no respecting set exists. By using appropriate data structures, this can be implemented in $\mathcal{O}(|E(\mathcal{N})|)$ time.

It remains to decide whether there exists a valid set of anchors F' for F. In particular, for each highest edge $uv \in F$, we need to choose one outgoing edge uv' with $c(uv') \in C$ to add to F' such that $c(e_1) \neq c(e_2)$ for each pair $e_1, e_2 \in F'$. This can be done by reducing to an instance of Perfect Matching, as follows. Construct a bipartite graph G with vertex set $F_h \cup C$, where F_h corresponds to the set of highest edges in F. For each $e \in F_h$ and $c \in C$, add an edge $\{e, c\}$ to G if e has a sibling edge e' with $c(e') = c$. Now, F is an (x, C)-respecting set if and only if G has a perfect matching covering F_h. Perfect Matching can be solved in $\mathcal{O}(|E(G)| \cdot \sqrt{|V(G)|})$ time with the famous Hopcroft-Karp Algorithm [19]. For each edge in $\mathcal{N}$, there is at most one edge in G. Thus, the overall running time is $\mathcal{O}(|E(\mathcal{N})| \cdot \sqrt{N})$. ◀

With respecting sets defined, we now formally define perfect triples. A triple (A, χ_1, χ_2), consisting of a set of taxa $A \subseteq X$, and mappings $\chi_1, \chi_2 : A \to 2^{[N]}$, is *perfect*, if

- the sets $\chi_i(x)$ and $\chi_i(y)$ are pairwise disjoint for $x, y \in A$ and $i \in \{1, 2\}$,
- the sets $\chi_1(x_i)$ and $\chi_2(x_j)$ are pairwise disjoint for $x_i, x_j \in A$ and $i \leq j$, and
- for each $x \in A$ there is a set of respecting edges $F_{x,\chi_2(x)} \subseteq E(\mathcal{N})$ with $\chi_1(x) = c(F_{x,\chi_2(x)})$.

To give some intuition behind the notion of a perfect triple, we observe that the existence of a perfect triple (A, χ_1, χ_2) provides a lower bound on the phylogenetic diversity of $(X \setminus A)$. The key idea is that as we remove the elements of A from X, one at a time, the set of edges lost with the removal of each $x \in A$ is a subset of the edges in $F_{x,\chi_2(x)}$.

▶ **Lemma 3.6 (⋆).** *$PD_{\mathcal{N}}(X \setminus A) \geq \omega(E(\mathcal{N})) - \sum_{x \in A} \omega(F_{x,\chi_2(x)})$ for every perfect triple (A, χ_1, χ_2).*

Having these definitions, we now define a colored problem, which we use as an auxiliary problem for solving Theorem 3.1. In ex-N-colored-Map-W-PDD, besides the usual input of Map-Weighted-PDD $(\mathcal{N}, \mathcal{F}, k, D)$ – where γ is a weighting of $\mathcal{F}$ – we are given a mapping $c : E(\mathcal{N}) \to [N]$ of a color per edge. We ask whether there exists a set of taxa $A \subseteq X$ of size *at least* $\overline{k}$ and mappings $\chi_1, \chi_2 : A \to 2^{[N]}$ such that $X \setminus A$ is γ-viable, the triple (A, χ_1, χ_2) is perfect, and $\sum_{x \in A} \omega(F_{x,\chi_2(x)}) \leq \overline{D}$.

Note that the existence of a perfect triple (A, χ_1, χ_2) does not imply that $X \setminus A$ is γ-viable.

▶ **Lemma 3.7 (⋆).** *Given a tree-extension $T_{\mathcal{F}}$ of the food web, we can solve instances of* ex-N-colored-Map-W-PDD *in $\mathcal{O}(5^N \cdot 2^{\mathrm{sw}_{\mathcal{F}}} \cdot n \cdot (N \cdot \overline{k} + |E(\mathcal{N})|))$ time.*

The intuition behind Lemma 3.7 is as follows: We consider the tree extension of the food web with a dynamic program bottom up. At each vertex v, we determine whether there exists a perfect triple (A, χ_1, χ_2) satisfying certain conditions, where A is a subset of $T_{\mathcal{F}}^{(v)}$, the set of vertices descended from v in $T_{\mathcal{F}}$. We want that $X \setminus A$ is γ-viable; to help determine this we keep track of a subset of edges $\Phi \subseteq GW(v)$, and require that $T_{\mathcal{F}}^{(v)} \setminus A$ is Φ-part-γ-viable.

It remains to show how to reduce instances of Map-Weighted-PDD to instances of ex-N-colored-Map-W-PDD. This is done using standard color-coding techniques.

Proof of Theorem 3.1.

Reduction. Let $\mathcal{I} = (\mathcal{N}, \mathcal{F}, k, D)$ be an instance of Map-Weighted-PDD. If $\mathcal{N}$ is a tree, then set N to $4k - 2$ and otherwise to $2D$.

Arbitrarily order the edges $e_1, \ldots, e_q$ of $\mathcal{N}$. We may assume $q > N$, as otherwise, we can consider a single instance of ex-q-colored-Map-W-PDD. Let $\mathcal{H}$ be a (q, N)-perfect hash family. For every $f \in \mathcal{H}$ we define a coloring c_f by $c_f(e_j) = f(j)$ for $j \in [q]$ and let $\mathcal{I}_f = (\mathcal{N}, \mathcal{F}, k, D, c_f)$ be the corresponding instance of ex-N-colored-Map-W-PDD. Solve every instance $\mathcal{I}_f$, and return `yes` if and only if $\mathcal{I}_f$ is a `yes`-instance for some $f \in \mathcal{H}$.

Correctness. The proof of the correctness and running time is deferred to the appendix. ◀

4 Parameter $k + \mathrm{sw}_{\mathcal{F}} + \delta + h$

By Theorem 2.5, Map-ε-PDD and Map-1-PDD are W[2]-hard with respect to $k + h$, even if the food web does not contain edges. In the following, we therefore add the maximum in-degree of a reticulation δ as a parameter and show that even the more general problem Map-Weighted-PDD is FPT with respect to $k + \mathrm{sw}_{\mathcal{F}} + h + \delta$.

To do this, we consider a parameter H generalizing the height of a tree. H is the maximum number of tree edges that, in $\mathcal{N}$, are on a path from the root to any taxon. We observe that in a phylogenetic tree, H is the height of the tree minus one. Next, we prove bounds on the value of H in a phylogenetic network.

▶ **Lemma 4.1** (⋆). *$h_t \leq H$ and $H \leq \delta^{h_r} \cdot h_t \leq \delta^h$.*

In this section we prove the following.

▶ **Theorem 4.2.** *Given a tree extension $T_{\mathcal{F}}$ of the food web with scanwidth $\mathrm{sw}_{\mathcal{F}}$,* Map-Weighted-PDD *can be solved in $\mathcal{O}(2^{2.443 \cdot kH + \mathrm{sw}_{\mathcal{F}} + \mathcal{O}(\log^2(kH))} \cdot \mathrm{sw}_{\mathcal{F}} \cdot n \cdot |E(\mathcal{N})|^2 \cdot \log |E_T(\mathcal{N})|)$ time.*

As Map-ε-PDD and Map-1-PDD are special cases of Map-Weighted-PDD, this result transfers to them.

▶ **Corollary 4.3.** Map-ε-PDD *and* Map-1-PDD *are* FPT *with respect to $k + \mathrm{sw}_{\mathcal{F}} + h + \delta$.*

First we define some objects necessary in this chapter. For a a set A of taxa and a mapping $\chi : X \to 2^{[k \cdot H]}$, we say that tuple (A, χ) is *colorful* if $\chi(x) \cap \chi(y) \neq \emptyset$ for $x, y \in A$ implies $x = y$.

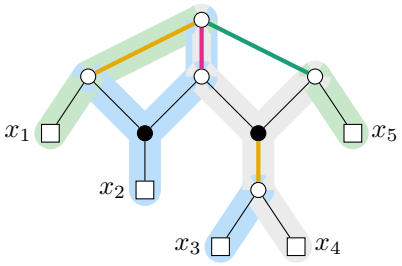
▶ **Definition 4.4.** *Fix a mapping $c : E_T(\mathcal{N}) \to [k \cdot H]$. For a taxon $x \in X$ and a mapping $\chi : X \to 2^{[k \cdot H]}$, A set of edges $F \subseteq E(\mathcal{N})$ is* (x, χ)-suitable, *if*

- *$c(e) \in \chi(x)$ for each edge $e \in F \cap E_T(\mathcal{N})$,*
- *$c(e_1) \neq c(e_2)$ for each pair $e_1, e_2 \in F \cap E_T(\mathcal{N})$, and*
- *for each edge $uv \in F$, there is a path from v to x in $\mathcal{N}[F]$.*

When (x, χ) is clear from the context we say an (x, χ)-respecting set is respecting.

Figure 2 shows an example. Note that a respecting set may contain reticulation edges.

To prove Theorem 4.2, we first show how to solve a colored variant of Map-Weighted-PDD. In kH-colored-Map-W-PDD, besides the usual input $(\mathcal{N}, \mathcal{F}, k, D)$, we are given a mapping $c : E_T(\mathcal{N}) \to [k \cdot H]$ of a color per tree edge. We ask whether a γ-viable set of taxa $S \subseteq X$ of size at most k and a mapping $\chi : S \to 2^{[k \cdot H]}$ exist such that (S, χ) is colorful and for each $x \in S$ there is a set of suitable edges $F_{x,\chi}$ and $\sum_{x \in S} \omega(F_{x,\chi}) \geq D$.



	$\chi(x)$
x_1	{●}
x_2	{●}
x_3	{●, ●}
x_4	{●, ●}
x_5	$\emptyset$

Figure 2 A hypothetical network on 3 colors {●, ●, ●}. For each taxon the (x, χ)-suitable set $F_{x,\chi}$ is indicated in ○, ○, or ○. In this network, H takes the value of 3 – there are three tree edges on paths to x_3 or x_4.

▶ **Lemma 4.5.** *Given an instance $\mathcal{I}$ of kH-colored-Map-W-PDD, a leaf x, and a mapping χ, a suitable edge set F with maximal value $\omega(F)$ can be computed in $\mathcal{O}(2^H \cdot |E(\mathcal{N})| \cdot (kH + |E(\mathcal{N})|))$ time.*

Because of this lemma, in the rest of the section, we simply will write $\omega(F_{x,\chi})$ when referring to the weight of a suitable edge set for x, which has maximal weight.

Proof.

Algorithm. Let E_x be the set of tree edges uv with $x \in \text{off}(v)$. For a taxon x and a mapping χ, iterate over all subsets F of E_x. If $uv \in F$ and u is a reticulation, then add all incoming edges of u to F, for each $uv \in F$. Add the edge incoming at x to F. Return the biggest value of a so computed F which is suitable.

Correctness. If the algorithm returns value d, then there is a suitable set F with $d = \omega(F)$.

Let F be a suitable set. The set $F_x := F \cap E_x$ appears in the iteration. The edge incoming at x is in F, as it is suitable. Reticulation edges have no color. Thus, adding reticulation edges leading to F keeps the set suitable. Consequently, a set F' with $F' \supseteq F$ has been considered by the algorithm.

Running Time. The set E_x has size at most H, by definition. All sets F are computed in $\mathcal{O}(2^H \cdot |E(\mathcal{N})|)$ time. Checking whether a set F is colorful can be done in $\mathcal{O}(kH + |E(\mathcal{N})|)$ time. ◀

▶ **Lemma 4.6.** *If (S, χ) is colorful, then any (x, χ)-suitable set $F_{x,\chi}$ and any (y, χ)-suitable set $F_{y,\chi}$ are pairwise disjoint for $x, y \in S$, $x \neq y$.*

Proof. Fix vertices $x, y \in S$ and an edge $uv \in F_{x,\chi}$. If uv is a tree edge, then $c(uv) \in \chi(x)$ and because $\chi(x)$ and $\chi(y)$ are disjoint, we conclude $uv \notin F_{y,\chi}$.

Now let uv be a reticulation edge. Let w be the unique *first* tree vertex on a path from v to x – or any other leaf. Then $c(e_w) \in \chi(x)$ for the edge e_w incoming at w. We conclude that $e_w \notin F_{y,\chi}$ and there is no path from v to y in $\mathcal{N}[F_{y,\chi}]$. Consequently $uv \notin F_{y,\chi}$.

If $v = x$, or the only taxon reachable from v is x, then $uv \notin F_{y,\chi}$. ◀

We write $F_{S,\chi}$ for the union $\bigcup_{x \in S} F_{x,\chi}$ for a set S. For a colorful set S we conclude with Lemma 4.6 that $PD_{\mathcal{N}}(S) \geq \omega(F_{S,\chi}) \geq D$.

We are now ready to prove how to solve instances of kH-colored-Map-W-PDD.

▶ **Lemma 4.7** (⋆). *Given an instance $\mathcal{I}$ of kH-colored-Map-W-PDD and a tree-extension $T_{\mathcal{F}}$ of the food web, we can solve $\mathcal{I}$ in $\mathcal{O}(2^{k \cdot H + \text{sw}_{\mathcal{F}}} \cdot k^4 \cdot H^2 \cdot \text{sw}_{\mathcal{F}} \cdot n \cdot |E(\mathcal{N})|^2)$ time.*

The intuition behind Lemma 4.7 is as follows: We consider the tree extension of the food web bottom up in a dynamic programming algorithm. We track the existence of colorful tuples whose taxa are all below a given vertex of the tree extension. We index colorful tuples by the sets of colors used, as well as by the set of edges Φ for which the set of taxa is Φ-part-γ-viable, and we store the total weight of the suitable edge sets. We use the fact that the sets $\chi(x)$ and $\chi(y)$ have to be pairwise disjoint. Therefore, if a taxon x is saved and a set of colors has been assigned, these colors can be removed. We define $\chi(x)$ when selecting x.

It remains to show how to reduce instances of Map-Weighted-PDD to an instances of kH-colored-Map-W-PDD. For this, we use perfect hash families. (Definition 2.1)

Proof of Theorem 4.2.

Reduction. Let $\mathcal{I} = (\mathcal{N}, \mathcal{F}, k, D)$ be an instance of Map-Weighted-PDD.

Arbitrarily order the tree edges $e_1, \ldots, e_q$ of $\mathcal{N}$. We may assume $q > k \cdot H$. Let $\mathcal{H}$ be a $(q, k \cdot H)$-perfect hash family. For every $f \in \mathcal{H}$ we define a coloring c_f by $c_f(e_j) = f(j)$ for $j \in [q]$ and let $\mathcal{I}_f = (\mathcal{N}, \mathcal{F}, k, D, c_f)$ be the corresponding instance of kH-colored-Map-W-PDD. Now, solve instance $\mathcal{I}_f$, and return `yes` if and only if $\mathcal{I}_f$ is a `yes`-instance for some $f \in \mathcal{H}$.

Correctness. For any set $E' \subseteq E_T(\mathcal{N})$ of edges with a size of at most $k \cdot H$, there is a function $f \in \mathcal{H}$ such that $c_f(E')$ contains each color at most once.

Now, let $\mathcal{I}$ be a `yes`-instance of Map-Weighted-PDD with solution $S = \{x_1, \ldots, x_k\} \subseteq X$. Further, let $E_T^{(1)}$ be the set of tree edges on paths from the root to x_1 in $\mathcal{N}$, and for $i \in [k-1]$ let $E_T^{(i+1)}$ be the set of tree edges on paths to x_{i+1} which are not in $E_T^{(i)}$. We define $E_T(S)$ as the union of these sets. By definition of H, each set $E_T^{(i)}$ has a size of at most H. By definition of perfect hash families, there is some $f \in \mathcal{H}$, such that c_f is injective on $E_T(S)$. Taking $\chi(x_i) = c_f(E_T^{(i)})$, we conclude that (S, χ) is colorful and $\omega(F_{S,\chi}) = PD_{\mathcal{N}}(S) \geq D$. Thus, (S, χ) is a solution of the `yes`-instance $\mathcal{I}_f$ of kH-colored-Map-W-PDD.

Conversely, whenever (S, χ) is a solution for instance $\mathcal{I}_f$, then S is also a solution for $\mathcal{I}$.

Running Time. The construction of $\mathcal{H}$ takes $e^{kH}(kH)^{\mathcal{O}(\log kH)} \cdot q \log q$ time, and for each $f \in \mathcal{H}$ the construction of instance $\mathcal{I}_f$ of kH-colored-Map-W-PDD takes time linear in $|\mathcal{I}|$. By Lemma 4.7, instances of kH-colored-Map-W-PDD can be solved in $\mathcal{O}(2^{k \cdot H + \mathrm{sw}_{\mathcal{F}}} \cdot k^4 \cdot H^2 \cdot \mathrm{sw}_{\mathcal{F}} \cdot n \cdot |E(\mathcal{N})|^2)$ time, and the number of instances is $|\mathcal{H}| = e^{kH}(kH)^{\mathcal{O}(\log kH)} \cdot \log q$.

Thus, the total running time is $\mathcal{O}(e^{kH}(kH)^{\mathcal{O}(\log kH)} \log q \cdot (q + 2^{k \cdot H + \mathrm{sw}_{\mathcal{F}}} \cdot k^4 \cdot H^2 \cdot \mathrm{sw}_{\mathcal{F}} \cdot n \cdot |E(\mathcal{N})|^2))$. This simplifies to $\mathcal{O}((2e)^{kH} \cdot 2^{\mathrm{sw}_{\mathcal{F}} + \mathcal{O}(\log^2(kH))} \cdot \mathrm{sw}_{\mathcal{F}} \cdot n \cdot |E(\mathcal{N})|^2 \cdot \log |E_T(\mathcal{N})|)$. ◀

By Theorem 2.4c, Map-Weighted-PDD is not FPT with respect to $k + h$. But this differs when we add $\mathrm{sw}_{\mathcal{F}}$ as a parameter. In a phylogenetic tree, the value of H is h.

▶ **Corollary 4.8.** Weighted-PDD *is* FPT *with respect to* $k + h + \mathrm{sw}_{\mathcal{F}}$ *and can be solved in* $\mathcal{O}(2^{2.443 \cdot kh + \mathrm{sw}_{\mathcal{F}} + \mathcal{O}(\log^2(kh))} \cdot \mathrm{sw}_{\mathcal{F}} \cdot n \cdot |E(\mathcal{N})|^2 \cdot \log |E_T(\mathcal{N})|)$ *time.*

Map-ε-PDD with respect to $k + \delta + h$

By Theorem 4.2, Map-Weighted-PDD is FPT with respect to $k + \mathrm{sw}_{\mathcal{F}} + \delta + h$ and therefore also Map-ε-PDD and Map-1-PDD. If any of the four parameters is dropped then Map-1-PDD is W[1]-hard, as pointed out in Table 2.

While the first three hardness results hold for both problems, the last explicitly only shows hardness for Map-1-PDD. We expect this hardness not to hold for Map-ε-PDD, but believe that with an approach similar to the one presented in [21] to show that ε-PDD is FPT with respect to $k + h$, one can also show that Map-ε-PDD is FPT when parameterized

Table 2 MAP-1-PDD is W[1]-hard if one of the parameters of $k + \mathrm{sw}_{\mathcal{F}} + \delta + h$ is dropped.

$\mathrm{sw}_{\mathcal{F}} + \delta + h$	$k + \mathrm{sw}_{\mathcal{F}} + \delta$	$k + \mathrm{sw}_{\mathcal{F}} + h$	$k + \delta + h$
Theorem 2.3,[9]	Corollary 2.6	Theorem 2.5,[4, 16]	Theorem 2.4c,[28]

with $k + \delta + h$. Unfortunately, the proof given in [21] has an incorrect lemma. An example of the error is explained in greater detail in this paper's Arxiv version [18]. We still believe both of the claims, the one [21] and following, to hold.

▶ **Conjecture 4.9.** MAP-ε-PDD *is* FPT *when parameterized with* $k + \delta + h$.

5 Discussion

In this paper, we made the approach to combine the natural problems of maximizing phylogenetic diversity in a network with viability constraints given through a food web. We defined the problem MAP-WEIGHTED-PDD and its special cases MAP-ε-PDD and MAP-1-PDD. We provided several FPT algorithms for these problems and even presented a complete complexity dichotomy by showing for which combination of parameters of k, $\overline{D}$, $\mathrm{sw}_{\mathcal{F}}$, δ, and h, the three problems are in FPT and for which they are W[1]-hard.

Still, several questions remain open. The most obvious is whether Conjecture 4.9 holds.

Further, in Section 3, we showed that MAP-WEIGHTED-PDD is FPT with respect to $\overline{D} + \mathrm{sw}_{\mathcal{F}}$. We showed that this approach is sufficient to show that WEIGHTED-PDD is FPT with respect to the smaller parameter $\overline{k} + \mathrm{sw}_{\mathcal{F}}$. Yet, it is unclear whether MAP-WEIGHTED-PDD admits an FPT algorithm for this parameter.

Another major open question, already pointed out in [21], is whether ε-PDD and 1-PDD are FPT when parameterized with k. This question even remains open if the food web is restricted to an out-tree – or if any vertex in $\mathcal{F}$ has a degree of at most 1.

In this paper, we observed the *All-Paths-PD* measure in phylogenetic networks as a measure of phylogenetic diversity. This measure is computationally the least challenging, but probably not the best measure in capturing phylogenetic diversity. In recent years, several other measures have been proposed [4, 31, 32, 34]. We wonder if there are also efficient algorithms for these problems that respect the viability of the selected set of taxa.

References

1 Noga Alon, Raphael Yuster, and Uri Zwick. Color-Coding. *Journal of the Association for Computing Machinery (JACM)*, 42(4):844–856, 1995. doi:10.1145/210332.210337.

2 Vincent Berry, Celine Scornavacca, and Mathias Weller. Scanning phylogenetic networks is NP-hard. In *Proceedings of the 46th International Conference on Current Trends in Theory and Practice of Informatics (SOFSEM 2020)*, pages 519–530. Springer, 2020.

3 Magnus Bordewich and Charles Semple. Nature Reserve Selection Problem: A Tight Approximation Algorithm. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 5(2):275–280, 2008. doi:10.1109/TCBB.2007.70252.

4 Magnus Bordewich, Charles Semple, and Kristina Wicke. On the complexity of optimising variants of phylogenetic diversity on phylogenetic networks. *Theoretical Computer Science*, 917:66–80, 2022. doi:10.1016/J.TCS.2022.03.012.

5 Gerardo Ceballos and Paul R. Ehrlich. Mutilation of the tree of life via mass extinction of animal genera. *Proceedings of the National Academy of Sciences*, 120(39):e2306987120, 2023.

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 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.

8 Daniel P. Faith. Conservation evaluation and Phylogenetic Diversity. *Biological Conservation*, 61(1):1–10, 1992.

9 Beáta Faller, Charles Semple, and Dominic Welsh. Optimizing Phylogenetic Diversity with Ecological Constraints. *Annals of Combinatorics*, 15(2):255–266, 2011.

10 Mariana Napolitano Ferreira. Conservation priorities mapping – A first step toward building area-based strategies. *Frontiers in Science*, 2:1440501, 2024.

11 Niels Holtgrefe. Computing the Scanwidth of Directed Acyclic Graphs. Master's thesis, Delft University of Technology, 2023.

12 Niels Holtgrefe, Jannik Schestag, and Norbert Zeh. Limits of Kernelization and Parametrization for Phylogenetic Diversity with Dependencies. Manuscript in preparation, 2025.

13 Niels Holtgrefe, Leo van Iersel, and Mark Jones. Exact and Heuristic Computation of the Scanwidth of Directed Acyclic Graphs. *arXiv preprint arXiv:2403.12734*, 2024. doi:10.48550/arXiv.2403.12734.

14 Niels Holtgrefe, Leo van Iersel, Ruben Meuwese, Yuki Murakami, and Jannik Schestag. PANDA: Maximizing All-Path Phylogenetic Diversity in Networks. Manuscript in preparation, 2025.

15 Daniel H. Huson and David Bryant. Application of Phylogenetic Networks in Evolutionary Studies. *Molecular biology and evolution*, 23(2):254–267, 2006.

16 Mark Jones and Jannik Schestag. How Can We Maximize Phylogenetic Diversity? Parameterized Approaches for Networks. In *Proceedings of the 18th International Symposium on Parameterized and Exact Computation (IPEC 2023)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.IPEC.2023.30.

17 Mark Jones and Jannik Schestag. Maximizing Phylogenetic Diversity under Time Pressure: Planning with Extinctions Ahead. *arXiv preprint arXiv:2403.14217*, 2024. doi:10.48550/arXiv.2403.14217.

18 Mark Jones and Jannik Schestag. Parameterized Algorithms for Diversity of Networks with Ecological Dependencies. *arXiv preprint*, 2025. arXiv:2510.09512.

19 Marek Karpiński and Wojciech Rytter. *Fast Parallel Algorithms for Graph Matching Problems*. Oxford University Press, 1998.

20 Christian Komusiewicz and Jannik Schestag. A Multivariate Complexity Analysis of the Generalized Noah's Ark Problem. In *Proceedings of the 19th Cologne-Twente Workshop on Graphs and Combinatorial Optimization*, pages 109–121. Springer, 2023.

21 Christian Komusiewicz and Jannik Schestag. Maximizing Phylogenetic Diversity under Ecological Constraints: A Parameterized Complexity Study. In *Proceedings of the 44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2024)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.FSTTCS.2024.28.

22 Bui Quang Minh, Steffen Klaere, and Arndt von Haeseler. Phylogenetic Diversity within Seconds. *Systematic Biology*, 55(5):769–773, October 2006. doi:10.1080/10635150600981604.

23 Vincent Moulton, Charles Semple, and Mike Steel. Optimizing phylogenetic diversity under constraints. *Journal of Theoretical Biology*, 246(1):186–194, 2007.

24 Moni Naor, Leonard J. Schulman, and Aravind Srinivasan. Splitters and near-optimal Derandomization. *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pages 182–191, 1995. doi:10.1109/SFCS.1995.492475.

25 Fabio Pardi and Nick Goldman. Species Choice for Comparative Genomics: Being Greedy Works. *PLoS Genetics*, 1(6):e71, 2005.

26 Fabio Pardi and Nick Goldman. Resource-Aware Taxon Selection for Maximizing Phylogenetic Diversity. *Systematic Biology*, 56(3):431–444, 2007.

27 Stuart L. Pimm, Clinton N. Jenkins, Robin Abell, Thomas M. Brooks, John L. Gittleman, Lucas N. Joppa, Peter H. Raven, Callum M. Roberts, and Joseph O. Sexton. The biodiversity of species and their rates of extinction, distribution, and protection. *Science*, 344(6187):1246752, 2014.

28 Jannik Schestag. Weighted Food Webs Make Computing Phylogenetic Diversity So Much Harder. *arXiv preprint*, 2025. `arXiv:2510.05911`.

29 Andreas Spillner, Binh T. Nguyen, and Vincent Moulton. Computing Phylogenetic Diversity for Split Systems. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 5(2):235–244, 2008. `doi:10.1109/TCBB.2007.70260`.

30 Mike Steel. Phylogenetic Diversity and the Greedy Algorithm. *Systematic Biology*, 54(4):527–529, 2005.

31 Leo van Iersel, Mark Jones, Jannik Schestag, Celine Scornavacca, and Mathias Weller. Average-Tree Phylogenetic Diversity of Networks. In *25th International Workshop on Algorithms in Bioinformatics (WABI 2025)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. `doi:10.4230/LIPIcs.WABI.2025.15`.

32 Leo van Iersel, Mark Jones, Jannik Schestag, Celine Scornavacca, and Mathias Weller. Phylogenetic Network Diversity Parameterized by Reticulation Number and Beyond. In *Proceedings of the 23rd RECOMB International Workshop on Comparative Genomics (RECOMB-CG 2025)*, Seoul, Republic of Korea, 2025.

33 Mark Vellend, William K. Cornwell, Karen Magnuson-Ford, and Arne Ø. Mooers. Measuring phylogenetic biodiversity, 2011.

34 Kristina Wicke and Mareike Fischer. Phylogenetic diversity and biodiversity indices on phylogenetic networks. *Mathematical Biosciences*, 298:80–90, 2018.

A Appendix – Omitted proofs

Proof of Theorem 3.1.

Reduction. Let $\mathcal{I} = (\mathcal{N}, \mathcal{F}, k, D)$ be an instance of Map-Weighted-PDD. If $\mathcal{N}$ is a tree, then set N to $4\overline{k} - 2$ and otherwise to $2\overline{D}$.

Arbitrarily order the edges $e_1, \dots, e_q$ of $\mathcal{N}$. We may assume $q > N$, as otherwise, we can consider a single instance of ex-q-colored-Map-W-PDD. Let $\mathcal{H}$ be a (q, N)-perfect hash family. For every $f \in \mathcal{H}$ we define a coloring c_f by $c_f(e_j) = f(j)$ for $j \in [q]$ and let $\mathcal{I}_f = (\mathcal{N}, \mathcal{F}, k, D, c_f)$ be the corresponding instance of ex-N-colored-Map-W-PDD. Solve every instance $\mathcal{I}_f$, and return `yes` if and only if $\mathcal{I}_f$ is a `yes`-instance for some $f \in \mathcal{H}$.

Correctness. For any subset of edges E' with $|E'| \leq N$, there is some $f \in \mathcal{H}$ such that c_f is injective on E' by Definition 2.1.

Let $\mathcal{I}_f$ be a `yes`-instance of ex-N-colored-Map-W-PDD. Thus, there is a perfect triple (A, χ_1, χ_2) such that $X \setminus A$ is γ-viable, A has a size of at least $\overline{k}$ and for each $x \in A$ there is a set of respecting edges $F_{x,\chi_2(x)} \subseteq E(\mathcal{N})$ with $\chi_1(x) = c(F_{x,\chi_2(x)})$ and $\sum_{x \in A} \omega(F_{x,\chi_2(x)}) \leq \overline{D}$. We show that $S := X \setminus A$ is a solution for instance $\mathcal{I}$. By definition, S is γ-viable and has a size of $|X| - |A| \leq |X| - \overline{k} = k$. As (A, χ_1, χ_2) is perfect, $F_{x,\chi_2(x)}$ have pairwise disjoint colors for $x, y \in A$ and are therefore disjoint. We conclude with Lemma 3.6 that $PD_{\mathcal{N}}(S) \geq \omega(E(\mathcal{N})) - \sum_{x \in A} \omega(F_{x,\chi_2(x)}) \geq \omega(E(\mathcal{N})) - \overline{D} = D$.

Now, let $\mathcal{I}$ be a `yes`-instance of Map-Weighted-PDD with solution $S \subseteq X$. Let $A := X \setminus S$ and since we can assume that S has a size of k, the size of A is $\overline{k}$. Let $E_A \subseteq E(\mathcal{N})$ be the set of edges uv where $\text{off}(v) \subseteq A$. As $D \leq PD_{\mathcal{N}}(S) = \omega(E(\mathcal{N}) \setminus E_A)$, we conclude that $\omega(E_A) \leq \overline{D}$.

Now define F_{x_1} to be the edges uv in E_A with $x_1 \in \text{off}(v)$, and define F_{x_i} to be the edges uv in $E_A \setminus \bigcup_{j=1}^{i-1} F_{x_j}$ with $x_i \in \text{off}(v)$. We define sets F'_{x_i} as follows. For each edge $uv \in F_{x_i}$, for which no incoming edge of u is in F_{x_i}, we add an edge uw outgoing of u to F'_{x_i}, where uw is from $Z_i := ((E(\mathcal{N}) \setminus E_A) \cup \bigcup_{j=1}^{i-1} F_{x_i}) \setminus \bigcup_{j=1}^{i-1} F'_{x_i}$. If F_{x_i} contains more than one edge outgoing of u, adding one edge is sufficient.

▷ Claim A.1. The set Z_i contains an edge outgoing of u.

Proof. Assume first that $\bigcup_{j=1}^{i-1} F'_{x_i}$ contains an edge uv'. Without loss of generality, let $uv' \in F'_{x_t}$ and $\bigcup_{j=t+1}^{i-1} F'_{x_i}$ does not contain edges outgoing of u. Then there is an edge $uv_t \in F_{x_t}$ and by construction uv_t is not in F'_{x_j} for any $j \in [i]$. Consequently, the set Z_i contains uv_t.

Now, assume that $\bigcup_{j=1}^{i-1} F'_{x_i}$ does not contain edges outgoing of u. If $E(\mathcal{N}) \setminus E_A$ contains an edge e outgoing of u, then $e \in Z_i$. Otherwise, all edges incoming at u are in E_A. As F_{x_i} contains no edge incoming at u, these edges and at least one edges e outgoing of u have to be in F_{x_t} for some $t \in [i-1]$. We conclude $e \in Z_i$ and the set Z_i contains an edge outgoing of u. ◁

The sets F_{x_i} are constructed to be the respecting sets and F'_{x_i} are the auxiliary sets from the definition of respecting sets. We observe that E_A is the union of all F_{x_i}. Thus, if $\mathcal{N}$ is a tree, then $|E_A| \leq 2\overline{k} - 1$, as a forest with $\overline{k}$ leaves has at most $2\overline{k} - 1$ edges. If $\mathcal{N}$ is a network, then $|E_A| \leq \omega(E_A) = \omega(E(\mathcal{N})) - PD_{\mathcal{N}}(S) \leq \overline{D}$. As we added at most one edge to F'_{x_i} per edge of F_{x_i}, we conclude that the union U of all F_{x_i} and F'_{x_i} contains at most $4\overline{k} - 2$ if $\mathcal{N}$ is a tree, and $2\overline{D}$ edges if $\mathcal{N}$ is a network.

Consequently, there is some $f \in \mathcal{H}$, such that c_f is injective on U. We set $\chi_1(x_i) = c_f(F_{x_i})$ and $\chi_2(x_i) = c_f(F'_{x_i})$. By the construction we conclude that (A, χ_1, χ_2) is perfect, S is γ-viable, and $\sum_{i=1}^{\overline{k}} \omega(F_{x_i,\chi_2(x_i)}) = \sum_{i=1}^{\overline{k}} \omega(F_{x_i}) = \omega(E_A) \leq \overline{k}$. Thus, $\mathcal{I}_f$ is a `yes`-instance of EX-N-COLORED-MAP-W-PDD.

Running Time. The construction of $\mathcal{H}$ takes $e^N N^{\mathcal{O}(\log N)} \cdot q \log q$ time (Proposition 2.2), and for each $f \in \mathcal{H}$ the construction of instance $\mathcal{I}_f$ of EX-N-COLORED-MAP-W-PDD takes time linear in $|\mathcal{I}|$. By Lemma 4.7, solving instances of EX-N-COLORED-MAP-W-PDD takes $\mathcal{O}(5^N 2^{\text{sw}_{\mathcal{F}}} \cdot n \cdot (N^2 \cdot \overline{k}^2 + |E(\mathcal{N})|))$ time, and the number of instances is $|\mathcal{H}| = e^N N^{\mathcal{O}(\log N)} \cdot \log q$.

Thus, the total running time is $\mathcal{O}(e^N N^{\mathcal{O}(\log N)} \log q \cdot (q + 5^N 2^{\text{sw}_{\mathcal{F}}} \cdot n \cdot (N^2 \cdot \overline{k}^2 + E(\mathcal{N}))))$. This simplifies to $\mathcal{O}((5e)^N \cdot 2^{\text{sw}_{\mathcal{F}} + \mathcal{O}(\log^2(N))} \cdot n \cdot |E(\mathcal{N})| \cdot \log |E(\mathcal{N})|)$, as $\overline{k} \leq N$.

Inserting $2\overline{D}$ or $4\overline{k} - 2$ into N, respectively, gives the desired running times. ◀

Proof of Lemma 3.6. Recall that $x_1, \ldots, x_n$ is the ordering of X given by a depth-first traversal of $T_{\mathcal{F}}$. For the sake of notational convenience, assume that $A = \{x_1, \ldots, x_{|A|}\}$.

The intuitive idea behind our proof is to show that, as we kill each of the taxa $x_1, \ldots, x_{|A|}$ in order, the amount of diversity we lose by killing x_i is at most $\omega(F_{x_i,\chi_2(x_i)})$. To this end, define $\text{Ext}_{\mathcal{N}}(A') := E(\mathcal{N}) \setminus E(X \setminus A')$ for any $A' \subseteq X$. That is, $\text{Ext}_{\mathcal{N}}(A')$ is the set of edges in $\mathcal{N}$ with all offspring in A'.

We prove the following claim by induction. $\text{Ext}_{\mathcal{N}}(\{x_1, \ldots, x_s\}) \subseteq \bigcup_{i=1}^{s} \omega(F_{x_i,\chi_2(x_i)})$, for each $s \in [|A|]$. Note that by letting $s = |A|$, this implies $PD_{\mathcal{N}}(X \setminus A) = \omega_\Sigma(E(\mathcal{N}) \setminus \text{Ext}_{\mathcal{N}}(A)) = \omega_\Sigma(E(\mathcal{N})) - \omega_\Sigma(\text{Ext}_{\mathcal{N}}(A)) \geq \omega_\Sigma(E(\mathcal{N}) - \sum_{x \in A} \omega(F_{x,\chi_2(x)})$, as required.

For the base case, observe that $\text{Ext}_{\mathcal{N}}(\{x_1\})$ consists of the edges uv for which v has a path to x, and either $v = x$ or v is a reticulation. But the definition of a respecting set implies that all such edges are also in $F_{x_1,\chi_2(x_1)}$, and so $\text{Ext}_{\mathcal{N}}(\{x_1\}) \subseteq F_{x_1,\chi_2(x_1)}$.

For the inductive step, assume that $\text{Ext}_{\mathcal{N}}(\{x_1, \ldots, x_s\}) \subseteq \bigcup_{i=1}^{s} \omega(F_{x_i,\chi_2(x_i)})$ for some $s < |A|$, and now assume for a contradiction that $\text{Ext}_{\mathcal{N}}(\{x_1, \ldots, x_{s+1}\})$ is not a subset of $\bigcup_{i=1}^{s+1} \omega(F_{x_i,\chi_2(x_i)})$. Then consider a lowest edge $e = uv$ in $\text{Ext}_{\mathcal{N}}(\{x_1, \ldots, x_{s+1}\}) \setminus$

$\bigcup_{i=1}^{s+1} \omega(F_{x_i,\chi_2(x_i)})$. By definition edges outgoing of v are also in $\textsc{Ext}_{\mathcal{N}}(\{x_1,\ldots,x_{s+1}\})$, and as uv is a lowest edge, all edges outgoing of v are also in $\bigcup_{i=1}^{s+1} \omega(F_{x_i,\chi_2(x_i)})$. At least one child edge vw is not in $\textsc{Ext}_{\mathcal{N}}(\{x_1,\ldots,x_s\})$ (otherwise $uv \in \textsc{Ext}_{\mathcal{N}}(\{x_1,\ldots,x_s\}) \subseteq \bigcup_{i=1}^{s} \omega(F_{x_i,\chi_2(x_i)})$), and this edge vw must be in $F_{x_{s+1},\chi_2(x_{s+1})}$. Then as v has an edge incoming which is not in $F_{x_{s+1},\chi_2(x_{s+1})}$, it follows by definition of a respecting set that vw has a sibling edge vw' with $c(vw') \in \chi_2(s+1)$.

Note however that $vw' \notin \bigcup_{i=1}^{s+1} \omega(F_{x_i,\chi_2(x_i)})$, as $c(vw') \in \chi_2(x_{s+1})$, which is disjoint from $\chi_1(v_i)$ for each $i \leq s$, and $\chi_1(e) \in \chi_1(v_i)$ for each $e \in F_{x_i,\chi_2(x_i)}$. Therefore, vw' is also not in $\textsc{Ext}_{\mathcal{N}}(\{x_1,\ldots,x_{s+1}\})$, as otherwise uv was not a lowest edge in $\textsc{Ext}_{\mathcal{N}}(\{x_1,\ldots,x_{s+1}\}) \setminus \bigcup_{i=1}^{s+1} \omega(F_{x_i,\chi_2(x_i)})$. It follows that w', and therefore v, has a path to a leaf which is not in $\{x_1,\ldots,x_{s+1}\}$, contradicting the assumption that uv was not a lowest edge in $\textsc{Ext}_{\mathcal{N}}(\{x_1,\ldots,x_{s+1}\})$. ◀

Proof of Lemma 3.7.

Intuition. We consider the tree extension of the food web with a dynamic program bottom up. At each vertex v, we determine whether there exists a perfect triple (A,χ_1,χ_2) satisfying certain conditions, where A is a subset of $T_{\mathcal{F}}^{(v)}$, the set of vertices descended from v in $T_{\mathcal{F}}$. We want that $X \setminus A$ is γ-viable; to help determine this we keep track of a subset of edges $\Phi \subseteq GW(v)$, and require that $T_{\mathcal{F}}^{(v)} \setminus A$ is Φ-part-γ-viable.

Table Definition. For a set of taxa $Q \subseteq X$, sets of colors $C_1, C_2 \subseteq [N]$, a set of edges $\Phi \subseteq E(\mathcal{F})$ between Q and $X \setminus Q$, and an integer $\ell \in [\overline{k}]_0$, let $\mathcal{S}_{(Q,C_1,C_2,\Phi,\ell)}$ be the set of perfect triples (A,χ_1,χ_2) of a set $A \subseteq Q$ of size *at least* ℓ and mappings $\chi_1,\chi_2 : A \to 2^C$ with $\chi_1(A) = C_1$ and $\chi_2(A) \setminus \chi_1(A) \subseteq C_2$, for which $Q \setminus A$ is Φ-part-γ-viable.

We define a dynamic programming algorithm with table DP. For a vertex $v \in V(T_{\mathcal{F}})$, sets of colors $C_1, C_2 \subseteq [N]$, a set of edges $\Phi \subseteq GW(v)$, and an integer $\ell \in [k]_0$, we store in $\mathrm{DP}[v, C_1, C_2, \Phi, \ell]$ the *minimum* value of $\omega(F_{A,\chi_2}) := \sum_{x \in A} \omega(F_{x,\chi_2(x)})$ for any perfect triple $(A,\chi_1,\chi_2) \in \mathcal{S}_{(T_{\mathcal{F}}^{(v)},C_1,C_2,\Phi,\ell)}$. If v an internal vertex of $T_{\mathcal{F}}$ with children $w_1,\ldots,w_t$, then we define further auxiliary tables $\mathrm{DP}_i[v, C_1, C_2, \Phi, \Psi, \ell]$ analogously, with (S,χ_1,χ_2) considered from $\mathcal{S}_{(Q_i,C_1,C_2,P_i,\ell)}$ with $Q_i := \bigcup_{j=1}^{i} T_{\mathcal{F}}^{(w_j)}$ and $P_i := (\Phi \cup \Psi) \cap \left(\bigcup_{j=1}^{i} GW(w_j)\right)$; that is – only the first i children of v are considered. We require that the set of edges Ψ is either empty or $\mathrm{pred}^{(E)}(v)$.

Algorithm. We define the table in a bottom-up fashion. Let v be a leaf of $T_{\mathcal{F}}$. (That is a taxon without predators). Fix disjoint color sets $C_1, C_2 \subseteq [N]$. If $\ell > 1$, we store ∞ in $\mathrm{DP}[v, C_1, C_2, \Phi, \ell]$. If $\ell = 0$ and $\gamma_\Sigma(\Phi) \geq 1$, we store 0 in $\mathrm{DP}[v, C_1, C_2, \Phi, \ell = 0]$. Otherwise – if $\ell = 1$ or $\gamma_\Sigma(\Phi) < 1$ – we store $\min\{\omega(F_{v,C_2'}) \mid C_2' \subseteq C_2, c(F_{v,C_2'}) = C_1\}$ in $\mathrm{DP}[v, C_1, C_2, \Phi, \ell = 1]$.

Now, let v be an internal vertex of $T_{\mathcal{F}}$ with children $w_1,\ldots,w_t$ such that w_i comes before w_{i+1} in the depth-first traversal ordering of X, for $i \in [t-1]$. We define $\mathrm{DP}_1[v, C_1, C_2, \Phi, \Psi, \ell]$ to be $\mathrm{DP}[w_1, C_1, C_2, (\Phi \cup \Psi) \cap GW(w_1), \ell]$. To compute further values for $i \in [t-1]$, we use the following recurrence in which we define $\mathrm{DP}_{i+1}[v, C_1, C_2, \Phi, \Psi, \ell]$ to be

$$\min \quad \mathrm{DP}_i[v, C_1', C_2' \cup (C_1 \setminus C_1'), \Phi, \Psi, \ell'] \tag{2}$$
$$+ \mathrm{DP}[w_{i+1}, C_1 \setminus C_1', C_2 \setminus C_2', (\Phi \cup \Psi) \cap GW(w_{i+1}), \ell - \ell']. \tag{3}$$

Here, we take the minimum over all $C_1' \subseteq C_1, C_2' \subseteq C_2$ and $\ell' \in [\ell]_0$.

Finally, if $v = s_{\mathcal{F}}$ or $\gamma_\Sigma(\Phi \cap \mathrm{prey}^{(E)}(v)) \geq 1$, we set $\mathrm{DP}[v, C_1, C_2, \Phi, \ell]$ to be

$$\min\{\ \mathrm{DP}_t[v, C_1, C_2, \Phi, \mathrm{pred}^{(E)}(v), \ell]\ ;\ \mathrm{DP}_t'\ \}. \tag{4}$$

Here, DP'_t is $\min_{C'_2 \subseteq C_2}\ \mathrm{DP}_t[v, C_1 \setminus c(F_{v,C'_2}), C_2 \setminus C'_2, \Phi, \emptyset, \ell - 1] + \omega(F_{v,C'_2})$. Otherwise, we set $\mathrm{DP}[v, C_1, C_2, \Phi, \ell]$ to be DP'_t. Intuitively, DP'_t corresponds to the case that v is one of the taxa to be going extinct, while $\mathrm{DP}_t[v, C_1, C_2, \Phi, \mathrm{pred}^{(E)}(v), \ell]$ corresponds to the case that $\mathrm{DP}_t[v, C_1, C_2, \Phi, \mathrm{pred}^{(E)}(v), \ell]$ corresponds to the case that v is saved.

We return `yes` if $\mathrm{DP}[s_{\mathcal{F}}, C_1, C_2, \emptyset, \overline{k}] \leq \overline{D}$, for some $C_1, C_2 \subseteq [N]$ which are disjoint. Otherwise, we return `no`.

Correctness. Let us quickly consider the basic case. For a leaf v in $T_{\mathcal{F}}$, the set $T_{\mathcal{F}}^{(v)}$ only contains v and so $\mathcal{S}_{(T_{\mathcal{F}}^{(v)}, C_1, C_2, \Phi, \ell)}$ for $\ell > 1$ is empty. If $\ell = 0$, the only possible triples (A, χ_1, χ_2) in $\mathcal{S}_{(T_{\mathcal{F}}^{(v)}, C_1, C_2, \Phi, \ell)}$ have $A \subseteq \{v\}$, i.e. we must let v survive or go extinct. We can only let v survive if $\{v\}$ is Φ-part-γ-viable. Thus we can store 0 if $\gamma_\Sigma(\Phi) \geq 1$ and otherwise the minimal value of $\omega(F_{A,\chi_2})$ is just $\omega(F_{v,\chi_2(x)})$, and so we store the minimum weight of a set of respecting edges $F_{v,C}$ such that $c(F_{v,C}) = C_1$ and $C \subseteq C_2$.

To show the correctness of Recurrence (2) and Recurrence (4), we assume that each previous table entries are correct and we prove that if $\mathrm{DP}_i[v, C_1, C_2, \Phi, \Psi, \ell] = d$, respectively $\mathrm{DP}[v, C_1, C_2, \Phi, \ell] = d$, then there is a triple (A, χ_1, χ_2) with $\omega(F_{A,\chi_2}) = d$ from $\mathcal{S}_{(Q_i, C_1, C_2, P_i, \ell)}$, respectively $\mathcal{S}_{(T_{\mathcal{F}}^{(v)}, C_1, C_2, \Phi, \ell)}$. Afterward, we show that for each such triple, it holds that $\mathrm{DP}_i[v, C_1, C_2, \Phi, \Psi, \ell] \geq \omega(F_{A,\chi_2})$, respectively $\mathrm{DP}[v, C_1, C_2, \Phi, \ell] \geq \omega(F_{A,\chi_2})$.

We first show the correctness of Recurrence (2). Assume that $\mathrm{DP}_{i+1}[v, C_1, C_2, \Phi, \Psi, \ell] = d$. Then, by Recurrence (2), there is a $d^1 \in [d]_0$ and $C'_1 \subseteq C_1, C'_2 \subseteq C_2, \ell' \in [\ell]_0$ such that $\mathrm{DP}_i[v, C'_1, C'_2, \Phi, \Psi, \ell'] = d^1$ and $\mathrm{DP}[w_{i+1}, C_1 \setminus C'_1, (C_2 \setminus C'_2) \cup C'_1, (\Phi \cup \Psi) \cap GW(w_{i+1}), \ell - \ell'] = d - d^1 =: d^2$. Consequently, there is $(A^1, \chi_1^1, \chi_2^1)$ in $\mathcal{S}_{(Q_i, C'_1, C'_2 \cup (C_1 \setminus C'_1), P_i, \ell')}$ such that $\omega(F_{A^1,\chi_2^1}) = d^1$ and there is $(A^2, \chi_1^2, \chi_2^2) \in \mathcal{S}_{(T_{\mathcal{F}}^{(w_{i+1})}, C_1 \setminus C'_1, (C_2 \setminus C'_2), (\Phi \cup \Psi) \cap GW(w_{i+1}), \ell - \ell')}$ such that $\omega(F_{A^2,\chi_2^2}) = d^2$. As Q and $T_{\mathcal{F}}^{(w_{i+1})}$ are disjoint, so also A^1 and A^2 are disjoint. We therefore define a set $A := A^1 \cup A^2$, and mappings χ_i with $\chi_i(x) = \chi_i^j(x)$ for each taxon $x \in A^j$, $i, j \in \{1, 2\}$. Then $\omega(F_{A,\chi_2}) = \omega(F_{A^1,\chi_2^1}) + \omega(F_{A^2,\chi_2^2}) = d^1 + d^2 = d$. It remains to show that (A, χ_1, χ_2) is in $\mathcal{S}_{(Q_{i+1}, C_1, C_2, P_{i+1}, \ell)}$. Most of the conditions follow because A^1 and A^2 are disjoint and the axioms hold for the individual sets. The difficult part is to show that $\chi_1(x_i)$ and $\chi_2(x_j)$ are disjoint for $i \leq j$, in the case that $x_i \in A^1$ and $x_j \in A^2$. For this, we observe that $\chi_1^1(A^1) \subseteq C'_1 \subseteq C_1$ and $\chi_2^2(A^2) \subseteq C_2 \setminus C'_2 \subseteq C_2$, and these are disjoint.

Now assume (A, χ_1, χ_2) is in $\mathcal{S}_{(Q_{i+1}, C_1, C_2, P_{i+1}, \ell)}$. Let $A^1 := A \cap Q_i$ and let χ_i^1 be the restriction of χ_i to A^1, for $i \in \{1, 2\}$. Similarly let $A^2 := A \setminus Q_i$, and let χ_i^2 be the restriction of χ_i to A^2, for $i \in \{1, 2\}$. It is straightforward to check that $(A^1, \chi_1^1, \chi_2^1)$ is in $\mathcal{S}_{(Q_i, C'_1, C'_2 \cup (C_1 \setminus C'_1), P_i, \ell')}$ and $(A^2, \chi_1^2, \chi_2^2) \in \mathcal{S}_{(T_{\mathcal{F}}^{(w_{i+1})}, C_1 \setminus C'_1, (C_2 \setminus C'_2), (\Phi \cup \Psi) \cap GW(w_{i+1}), \ell - \ell')}$, where $C'_1 = \chi_1(A^1), C'_2 = \chi_2(A^1) \setminus C'_1$, and $\ell = |A^1|$.

We next show the correctness of Recurrence (4). Assume that $\mathrm{DP}[v, C_1, C_2, \Phi, \ell] = d$. By Recurrence (4), we have $\mathrm{DP}_t[v, C_1, C_2, \Phi, \mathrm{pred}^{(E)}(v), \ell] = d$ or $\min_{C'_2 \subseteq C_2}\ \mathrm{DP}_t[v, C_1 \setminus c(F_{v,C'_2}), C_2 \setminus C'_2, \Phi, \emptyset, \ell - 1] + \omega(F_{v,C'_2}) = d$. In the former case, there is (A, χ_1, χ_2) in $\mathcal{S}_{(Q_t, C_1, C_2, P_t, \ell)}$ with $\omega(F_{A,\chi_2}) = d$. We observe that (A, χ_1, χ_2) is also in in $\mathcal{S}_{(T_{\mathcal{F}}^{(v)}, C_1, C_2, \Phi, \ell)}$ which is sufficient for this case. in the latter case, fix C'_2 such that $\mathrm{DP}_t[v, C_1 \setminus c(F_{v,C'_2}), (C_2 \setminus C'_2) \cup c(F_{v,C'_2}), \Phi, \Psi = \emptyset, \ell - 1] + \omega(F_{v,C'_2}) = d$. Consequently, there is a triple (A, χ_1, χ_2) in $\mathcal{S}_{(Q_t, C_1 \setminus c(F_{v,C'_2}), (C_2 \setminus C'_2) \cup c(F_{v,C'_2}), P_t, \ell)}$. Consider $(A', \chi_1\, \chi'_2)$ with $A' := A \cup \{v\}$, χ'_i is χ_i on A with $\chi_2(v) = C'_2$ and $\chi'_1(v) := c(F_{v,C'_2})$. Clearly $\omega(F_{A',\chi_2}) = \omega(F_{A,\chi_2}) + \omega(F_{v,C'_2}) = d$. As $(T_{\mathcal{F}}^{(v)} \setminus A' = Q_t \setminus A$ and $Q_t \setminus A$ is P_t-part-γ-viable, we have also that $T_{\mathcal{F}}^{(v)} \setminus A'$ is ϕ-part-γ-viable

(note that the only edges in $P_t \setminus \phi$ are those incoming at v, which are not needed for a set not containing v.) It remains to show that (A', χ_1', χ_2') is perfect, assuming that (A, χ_1, χ_2) is perfect.

Note that v appears before any vertex in Q_t in the depth-first traversal of $T_{\mathcal{F}}$, and so we may assume v is the first element of A'. It is clear that $\chi_1'(v) = c(F_{v,C_2'})$ is disjoint from $C_1 \setminus c(F_{v,C_2'})$, and so $\chi_1'(x)$ and $\chi_1'(y)$ are disjoint for all $x, y \in A'$. As $\chi_2(y) \subseteq C_2 \setminus C_2' \cup (C_1 \setminus c(F_{v,C_2'}))$ for all $y \in A$ and C_1, C_2' are disjoint, we have that $\chi_2'(v) = C_2'$ and $\chi_2'(y)$ are disjoint. Therefore $\chi_2'(x)$ and $\chi_2'(y)$ are disjoint for all $x, y \in A'$. Similarly as $\chi_1'(v) = c(F_{v,C_2'})$ and $\chi_2(y) \subseteq C_2 \setminus C_2' \cup (C_1 \setminus c(F_{v,C_2'})$ for $y \in A$, we have that $\chi_1'(x_i)$ and $\chi_2'(x_j)$ are pairwise disjoint for all $x_i, x_j \in A'$ with $i < j$. The existence of $F_{x,C_2'}$ follows from $\mathrm{DP}_t' + \omega(F_{v,C_2'}) = d$.

On the converse, if (A, χ_1, χ_2) is in $\mathcal{S}_{(T_{\mathcal{F}}^{(v)}, C_1, C_2, \Psi, \ell)}$, then we can show by a case distinction and with arguments similar to the previous paragraph that $\mathrm{DP}[v, C_1, C_2, \Phi, \ell] \leq \omega(F_{A,\chi_2})$.

Running Time. By Lemma 3.5, we can compute $F_{x,C}$ or conclude that it does not exist for all $x \in X$ and $C \subseteq [N]$ in $\mathcal{O}(2^N \cdot \sqrt{N} \cdot n \cdot |E(\mathcal{N})|)$ time.

We observe that because C_1 and C_2 are disjoint, $T_{\mathcal{F}}$ is a tree and therefore any vertex has at most one parent, and the field Ψ can only take two values for a fixed vertex v.

In the basic cases, in Recurrence (2), and in Recurrence (4), we iterate over $C_2' \subseteq C_2$ and in Recurrence (2) we additionally iterate over $C_1' \subseteq C_1$. Any color $c \in [N]$ can therefore be in $[N] \setminus (C_1 \cup C_2), C_2 \setminus C_2', C_2', C_1 \setminus C_1'$ or in the case of Recurrence (2) also in C_1'.

Thus, all table entries can be computed within $\mathcal{O}(5^N 2^{\mathrm{sw}_{\mathcal{F}}} \cdot n \cdot (N \cdot \overline{k} + |E(\mathcal{N})|))$ time. ◀

Proof of Lemma 4.1. Recall h_r (and h_t) is the maximum number of reticulations (respectively tree vertices) on a path from the root to a leaf. For each path P from the root to a leaf we observe $|E(P) \cap E_T(\mathcal{N})| \leq H$ and as there is a path of containing h_t tree edges, we conclude $h_t \leq H$.

To see that $H \leq \delta^{h_r} \cdot h_t \leq \delta^h$, let $\mathcal{N}$ be a network maximizing the value of H for values of h_r, h_t, δ, and let $x \in X$ be a leaf for which the maximum number of tree vertices with a path to x is maximized. We show that there is no tree vertex below a reticulation on any path to x. Assume for a contradiction that there is an edge $rv_1 \in E(\mathcal{N})$ with $v_1 \in V_T(\mathcal{N})$ and $r \in V_R(\mathcal{N})$ and a path from v_1 to x. Let $u_1, \ldots, u_s$ be the parents of r and w a child of v_1 such that w has a path to x. Remove the edges $u_i r$ for $i \in [s]$, rv_1, and $v_1 w$, add vertices $v_2, \ldots, v_s$ with attached leaves, and add edges $u_i v_i$ for $i \in [s]$, $v_i r$ for $i \in [s]$ and rw. Observe that after this transformation, the values of h_r, h_t, and δ have not changed, but there are $s - 1$ further tree vertices with a path to x. As $s > 1$, this contradicts the maximality.

We may therefore assume that no tree vertices with a path to x are below a reticulation. We conclude that from the root at most δ^{h_r} different paths of length h_t of tree vertices can lead to a leaf. Figure 3 shows this scenario. ◀

Proof of Lemma 4.7.

Intuition. We consider the tree extension of the food web bottom up in a dynamic programming algorithm. We track the existence of colorful tuples whose taxa are all below a given vertex of the tree extension. We index colorful tuples by the sets of colors used, as well as by the set of edges Φ for which the set of taxa is Φ-part-γ-viable, and we store the total weight of the suitable edge sets. We use the fact that the sets $\chi(x)$ and $\chi(y)$ have to be pairwise disjoint. Therefore, if a taxon x is saved and a set of colors has been assigned, these colors can be removed. We define $\chi(x)$ when selecting x.

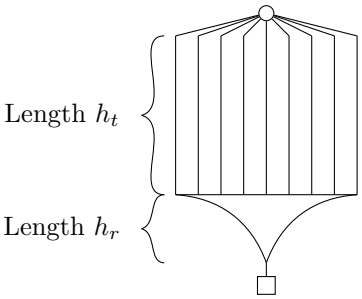


Figure 3 An example for Lemma 4.1 where H is maximized. This is the case if above the leaf an upside-down pyramid of reticulations is followed by δ^{h_r} paths of tree vertices of length h_t.

Table Definition. For a set of taxa $Q \subseteq X$, a set of colors $C \subseteq [k{\cdot}H]$, a set of edges $\Phi \subseteq E(\mathcal{F})$ between Q and $X \setminus Q$, and an integer $\ell \in [k]_0$, let $\mathcal{S}_{(Q,C,\Phi,\ell)}$ be the set of colorful tuples (S, χ) of a set $S \subseteq Q$ of size *at most* ℓ and a mapping $\chi : S \to 2^C$ with $\chi(S) = C$ such that S is Φ-part-γ-viable. We define a dynamic programming algorithm with table DP. For a vertex $v \in V(T_\mathcal{F})$, a set of colors $C \subseteq [k{\cdot}H]$, a set of edges $\Phi \subseteq GW(v)$, and an integer $\ell \in [k]_0$, we store in $\mathrm{DP}[v, C, \Phi, \ell]$ the *maximum* value $\omega(F_{S,\chi})$ of any tuple $(S, \chi) \in \mathcal{S}_{(T_\mathcal{F}^{(v)},C,\Phi,\ell)}$. The value of an optimal set can then be found in $\mathrm{DP}[s_\mathcal{F}, [k \cdot H], \emptyset, k]$. Is v an internal vertex of $T_\mathcal{F}$ with children $w_1, \ldots, w_t$, then we define further auxiliary tables $\mathrm{DP}_i[v, C, \Phi, \Psi, \ell]$ analogously, where tuples (S, χ) are considered from $\mathcal{S}_{(Q,C,P,\ell)}$ with $Q := \bigcup_{j=1}^{i} T_\mathcal{F}^{(w_j)}$ and $P := (\Phi \cup \Psi) \cap \left(\bigcup_{j=1}^{i} GW(w_j)\right)$; that is – only the first i children of v are considered. We require that the set of edges Ψ is either empty or $\mathrm{pred}^{(E)}(v)$.

Algorithm. We define the table in a bottom-up fashion. Let v be a leaf of $T_\mathcal{F}$. (That is a taxon without predators). In $\mathrm{DP}[v, C, \Phi, \ell]$, we store $\omega(F_{v,\chi})$ with $\chi(v) = C$ if $\gamma_\Sigma(\Phi) = 1$ and $\ell \geq 1$. Otherwise, we store 0.

Now let v be an internal vertex of $T_\mathcal{F}$ with children $w_1, \ldots, w_t$. We define $\mathrm{DP}_1[v, C, \Phi, \Psi, \ell]$ to be $\mathrm{DP}[w_1, C, (\Phi \cup \Psi) \cap GW(w_1), \ell]$. To compute further values for $i \in [t-1]$, we use the following recurrence in which we define $\mathrm{DP}_{i+1}[v, C, \Phi, \Psi, \ell]$ to be

$$\max_{C' \subseteq C, \ell' \in [\ell]_0} \mathrm{DP}_i[v, C', \Phi, \Psi, \ell'] + \mathrm{DP}[w_{i+1}, C \setminus C', (\Phi \cup \Psi) \cap GW(w_{i+1}), \ell - \ell']. \tag{5}$$

Finally, if $\ell \geq 1$ and ($v = s_\mathcal{F}$ or $\gamma_\Sigma(\Phi \cap \mathrm{prey}^{(E)}(v)) \geq 1$), we set $\mathrm{DP}[v, C, \Phi, \ell]$ to be

$$\max\{\ \mathrm{DP}_t[v, C, \Phi, \emptyset, \ell]\ ;\ \max_{C' \subseteq C}\ \mathrm{DP}_t[v, C \setminus C', \Phi, \mathrm{pred}^{(E)}(v), \ell - 1] + \omega(F_{v,\chi})\ \}. \tag{6}$$

Here, we use $\chi(v) := C'$. Otherwise, we set $\mathrm{DP}[v, C, \Phi, \ell]$ to be $\mathrm{DP}_t[v, C, \Phi, \emptyset, \ell]$.

We return `yes` if $\mathrm{DP}[s_\mathcal{F}, [k \cdot H], \emptyset, k] \geq D$. Otherwise, we return `no`.

Correctness. We only show the correctness of Recurrence (6) and omit to show the similar case in Recurrence (5) as well as the basic cases. Assume that DP stores the correct value for all children of v in $T_\mathcal{F}$ and DP_t stores the correct value for v. We show first that if $\mathrm{DP}[v, C, \Phi, \ell] = d$, then there is a tuple $(S, \chi) \in \mathcal{S}_{(T_\mathcal{F}^{(v)},C,\Phi,\ell)}$ and $\omega(F_{S,\chi}) = d$. Afterward, we show that $\mathrm{DP}[v, C, \Phi, \ell] \geq \omega(F_{S,\chi})$ for each tuple $(S, \chi) \in \mathcal{S}_{(T_\mathcal{F}^{(v)},C,\Phi,\ell)}$.

So assume that $\mathrm{DP}[v, C, \Phi, \ell] = d$. Consequently, we have $\mathrm{DP}_t[v, C, \Phi, \emptyset, \ell] = d$, or there is some $C' \subseteq C$ such that $\mathrm{DP}_t[v, C \setminus C', \Phi, \mathrm{pred}^{(E)}(v), \ell - 1] = d - \omega(F_{v,\chi})$ for $\chi(v) = C'$. In the former case, there is a colorful tuple (S, χ) with a set $S \subseteq \bigcup_{i=1}^{t} T_\mathcal{F}^{(w_i)} = T_\mathcal{F}^{(v)} \setminus \{v\}$ of size ℓ and a mapping χ such that $\omega(F_{S,\chi}) = d$ and S is $(\Phi \cup \emptyset)$-part-γ-viable. Consequently, S is

also Φ-part-γ-viable in $T_{\mathcal{F}}^{(v)}$. In the latter case, $\ell \geq 1$ and there is a colorful tuple (S, χ) with a set $S \subseteq T_{\mathcal{F}}^{(v)} \setminus \{v\}$ of size $\ell - 1$ and a mapping χ with $\chi(S) = C \setminus C'$, such that $\chi(v) = C'$ and $\omega(F_{S,\chi}) = d - \omega(F_{v,\chi})$. Thus, adding v to S yields the desired set.

Conversely, let $(S, \chi) \in \mathcal{S}_{(T_{\mathcal{F}}^{(v)}, C, \Phi, \ell)}$ be given. Assume first that v is not in S. Because S is Φ-part-γ-viable, we conclude $\mathrm{DP}[v, C, \Phi, \ell] = \mathrm{DP}_t[v, C, \Phi, \emptyset, \ell] \geq \omega(F_{S,\chi})$. Now, let v be in S. As S is Φ-part-γ-viable and $\mathrm{prey}^{(E)}(x) \cap GW(v) = \mathrm{prey}^{(E)}(x) \cap \Phi$ for each $x \in S$, the set $S \setminus \{v\}$ is $(\Phi \cup \mathrm{pred}^{(E)}(v))$-part-$\gamma$-viable. We conclude $(S \setminus \{v\}, \chi) \in \mathcal{S}_{(T_{\mathcal{F}}^{(v)} \setminus \{v\}, C, \Phi, \ell-1)}$ and therefore $\mathrm{DP}_t[v, C, \Phi, \mathrm{pred}^{(E)}(v), \ell - 1] \geq \omega(F_{S,\chi}) - \omega(F_{v,\chi})$ and further $\mathrm{DP}[v, C, \Phi, h] \geq \omega(F_{S,\chi})$.

Running Time. Observe that for any vertex v, only two values are possible for Ψ. Therefore, all tables have $\mathcal{O}(2^{k \cdot H + \mathrm{sw}_{\mathcal{F}}} \cdot nk)$ entries, together. By Recurrence (5), in time $\mathcal{O}(2^{k \cdot H} \cdot 2^{\mathrm{sw}_{\mathcal{F}}} \cdot k^2 \cdot \mathrm{sw}_{\mathcal{F}} \cdot (kH) \cdot n)$ all entries of DP_{i+1} can be computed, using convolutions. By Recurrence (6), we can compute $\mathrm{DP}[v, C, \Phi, \ell]$ in $\mathcal{O}(k \cdot H)$ time and $\omega(F_{v,\chi})$ needs to be computed once per vertex, which by Lemma 4.1 can be done in $\mathcal{O}(2^H \cdot |E(\mathcal{N})| \cdot (kH + |E(\mathcal{N})|))$ time. This leads to an overall running time of $\mathcal{O}(2^{k \cdot H + \mathrm{sw}_{\mathcal{F}}} \cdot k^4 \cdot H^2 \cdot \mathrm{sw}_{\mathcal{F}} \cdot n \cdot |E(\mathcal{N})|^2)$. ◀

Timeline Problems in Temporal Graphs: Vertex Cover vs. Dominating Set

Anton Herrmann
Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

Christian Komusiewicz
Institute of Computer Science, Friedrich Schiller University Jena, Germany

Nils Morawietz
Institute of Computer Science, Friedrich Schiller University Jena, Germany
LaBRI, Université de Bordeaux, Talence, France

Frank Sommer
Institute of Logic and Computation, TU Wien, Austria
Institute of Computer Science, Friedrich Schiller University Jena, Germany

Abstract

A temporal graph is a finite sequence of graphs, called snapshots, over the same vertex set. Many temporal graph problems turn out to be much more difficult than their static counterparts. One such problem is Timeline Vertex Cover (also known as MinTimeline$_\infty$), a temporal analogue to the classical Vertex Cover problem. In this problem, one is given a temporal graph $\mathcal{G}$ and two integers k and ℓ, and the goal is to cover each edge of each snapshot by selecting for each vertex at most k activity intervals of length at most ℓ each. Here, an edge uv in the ith snapshot is covered, if an activity interval of u or v is active at time i. In this work, we continue the algorithmic study of Timeline Vertex Cover and introduce the Timeline Dominating Set problem where we want to dominate all vertices in each snapshot by the selected activity intervals.

We analyze both problems from a classical and parameterized point of view and also consider partial problem versions, where the goal is to cover (dominate) at least t edges (vertices) of the snapshots. With respect to the parameterized complexity, we consider the temporal graph parameters vertex-interval-membership-width (vimw) and interval-membership-width (imw). We show that all considered problems admit FPT-algorithms when parameterized by $\text{vimw} + k + \ell$. This provides a smaller parameter combination than the ones used for previously known FPT-algorithms for Timeline Vertex Cover. Surprisingly, for $\text{imw} + k + \ell$, Timeline Dominating Set turns out to be easier than Timeline Vertex Cover, by also admitting an FPT-algorithm, whereas the vertex cover version is NP-hard even if $\text{imw} + k + \ell$ is constant. We also consider parameterization by combinations of n, the vertex set size, with k or ℓ and parameterization by t. Here, we show for example that both partial problems are fixed-parameter tractable for t which significantly improves and generalizes a previous result for a special case of Partial Timeline Vertex Cover with $k = 1$.

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

Keywords and phrases NP-hard problem, FPT-algorithm, interval-membership-width, Color coding

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.12

Related Version Some of the results of this work are also contained in the first author's Master's thesis [21].
Full Version: https://doi.org/10.48550/arXiv.2510.08124

Funding *Nils Morawietz*: Supported by the French ANR, project ANR-22-CE48-0001 (TEMPOGRAL).
Frank Sommer: Supported by the Alexander von Humboldt Foundation.

© Anton Herrmann, Christian Komusiewicz, Nils Morawietz, and Frank Sommer;
licensed under Creative Commons License CC-BY 4.0

1 Introduction

A crucial task in the management of wireless sensor networks is to monitor the network by selecting few dedicated sensors that can monitor themselves and other sensors which are close enough to have a direct wireless connection [27, 30]. In graph-theoretic terms, this is the famous NP-hard Dominating Set problem where we say that a vertex dominates itself and all its neighbors and the task is to select few vertices of the graph such that every vertex is dominated by some selected vertex. In some applications of sensor networks, the network may change over time. Then, instead of a static graph G the input would be a *temporal graph* $\mathcal{G}$, consisting of a set of *snapshots* G_i, each reflecting the connections at timestep i. Moreover, in such a scenario each sensor could carry out the surveillance only for a bounded duration, for example due to limited battery capacities. Thus, instead of selecting few sensors to monitor the network, we would like to select, for each sensor, an active time interval of bounded length ℓ, during which it monitors itself and all its current neighbors. To make the model more general, we may also allow for each sensor to select up to k such active intervals, for example because each sensor carries k batteries. By calling a collection of active intervals for all vertices of the graph a *k-activity ℓ-timeline*, the scenario described above corresponds to the following problem.

Timeline Dominating Set (Timeline DS)
Input: A temporal graph $\mathcal{G}$ and integers $k \geq 1, \ell \geq 0$.
Question: Is there a k-activity ℓ-timeline $\mathcal{T}$ which dominates all temporal vertices of $\mathcal{G}$?

For an example input and solution for Timeline DS, refer to Figure 1. As in the static Dominating Set problem, we may further generalize the problem to handle scenarios where we are not able to monitor the whole network over the full time period but instead we want to maximize the number of monitored sensors under the resource limitations.

Timeline Partial Dominating Set (Timeline PDS)
Input: A temporal graph $\mathcal{G}$ and integers $k \geq 1, \ell \geq 0, t \geq 0$.
Question: Is there a k-activity ℓ-timeline $\mathcal{T}$ which dominates at least t temporal vertices of $\mathcal{G}$?

In this work, we initialize the study of Timeline (Partial) Dominating Set in particular from a computational complexity perspective. As we show, Timeline (Partial) Dominating Set is NP-hard. To cope with this intractability, we consider mostly the parameterized complexity of the problem, with a particular focus on structural parameterizations of temporal graphs.

The idea to consider timelines in a temporal graph setting is not new: Rozenshtein et al. [28] introduced it in Timeline Vertex Cover,[1] the second main problem studied in this work.

[1] Rozenshtein et al. denote this problem as MinTimeline$_\infty$.

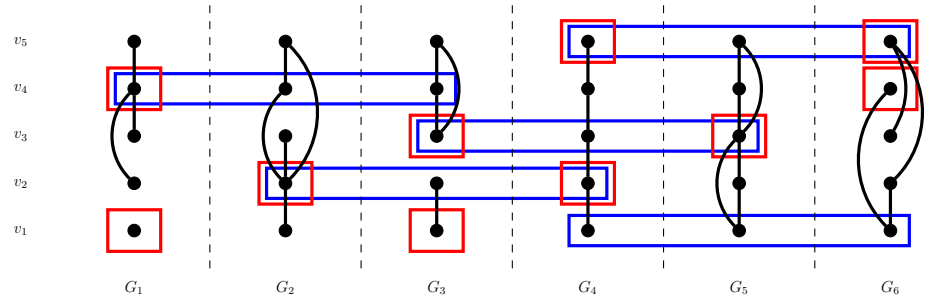


Figure 1 The short red boxes represent a 2-activity 0-timeline dominating all temporal vertices and the long blue boxes represent a 1-activity 2-timeline covering all temporal edges. Observe that the interval length is defined in such a way that an interval starting and ending in the same snapshot has length 0.

Timeline Vertex Cover (Timeline VC)
Input: A temporal graph $\mathcal{G}$ and integers $k \geq 1, \ell \geq 0$.
Question: Is there a k-activity ℓ-timeline $\mathcal{T}$ which covers all temporal edges of $\mathcal{G}$?

For an example input and solution for Timeline VC, refer again to Figure 1. The model behind Timeline VC is that edges in a temporal graph arise only when at least one of their endpoints is active and the task of the problem is to provide an explanation of all edges such that the vertices are only active a few times, each time only for a short duration [28]. Dondi et al. [12] later analyzed the partial variant of the problem.

Timeline Partial Vertex Cover (Timeline PVC)
Input: A temporal graph $\mathcal{G}$ and integers $k \geq 1, \ell \geq 0, t \geq 0$.
Question: Is there a k-activity ℓ-timeline $\mathcal{T}$ which covers at least t temporal edges of $\mathcal{G}$?

We continue the study of Timeline (Partial) Vertex Cover with two aims: First, some of the parameters considered in our study have not yet been studied for this problem. Second, we seek a comprehensive complexity overview for these two fundamental timeline problems, highlighting their similarities and differences.

Previous results. Rozenshtein et al. [28] showed that Timeline VC is NP-hard in general, but solvable in polynomial time for $k = 1$. Froese et al. [18] continued the algorithmic study of Timeline VC. They proved NP-hardness even if the input consists of at most three snapshots, $k = 2$ and $\ell = 0$ and studied the parameterized complexity with respect to different parameters. Froese et al. [18] showed fixed-parameter tractability for $n+k$ and W[1]-hardness for $n + \ell$. Herein, n is the number of vertices of the underlying graph which in turn is the static graph obtained by taking the union of all edge sets. Moreover, Timeline VC admits an XP-algorithm for n and is fixed-parameter tractable with respect to n if $\ell = 0$ [18]. Schubert [29] later analyzed the parameterized complexity of Timeline VC with respect to combinations of input parameters with structural parameters of the underlying graph. Finally, Dondi et al. [12] obtained two results for Timeline PVC with $k = 1$: First, it was

shown that this case is NP-hard, in contrast to Timeline VC. Second, Dondi et al. [12] gave FPT-algorithms for parameterization by the number of covered edges and by the number of uncovered edges.

Our results. We start off with a new hardness result for Timeline VC: We show that the problem remains NP-hard even if the total number of snapshots T, k, and ℓ are constant and additionally the maximum degree in every snapshot is at most one. Then, we show the NP-hardness of Timeline DS, again for the case that T, k, and ℓ are constant and the maximum snapshot degree is one.

We then study the parameterized complexity of the problems; an overview of the results is given in Table 1. As noted above, Timeline VC is fixed-parameter tractable with respect to $n + k$ [18]. Since n is a rather large parameter, Froese et al. [18] asked whether there are FPT-algorithms also for parameters that are smaller than n. We make progress in this direction by considering the two parameters *interval-membership-width* (imw) and *vertex-interval-membership-width* (vimw) which where introduced by Bumpus and Meeks [3]. The parameters imw and vimw are interesting in the sense that they are among the relatively few truly temporal structural graph parameters because they are sensitive to the order of the snapshots [3]. Informally, vimw can be defined as follows: We say the *lifetime* of a vertex is the time interval between its first and last non-isolated appearance in a snapshot. For a timestep i, the bag of i is the set of vertices whose lifetime contains timestep i, and vimw is the size of the largest bag of $\mathcal{G}$.

We first show that Timeline PVC is fixed-parameter tractable with respect to vimw $+k+\ell$. To further improve on this, we introduce a hierarchy of parameters with non-increasing size. The idea, inspired by the h-index of static graphs [16], is that a temporal graph may have only few large bags. In that case, we would rather parameterize by the size of the xth largest bag, denoted by vimw$[x]$, for some $x > 1$. With this definition, we have vimw = vimw$[1]$ and vimw$[x] \geq$ vimw$[x + 1]$ for all $x \in [T]$. We show that Timeline PVC is fixed-parameter tractable for vimw$[x] + k + \ell$ with $x = k \cdot (\ell + 1)$. Informally, this means that the larger k and ℓ get, the more large bags can be ignored for the parameter. We then consider Timeline (Partial) Dominating Set in the same setting. We first showing fixed-parameter tractability of Timeline PDS for vimw $+ k + \ell$. Then we show that for Timeline DS it can be improved to fixed-parameter tractability for vimw$[x] + k + \ell$ for $x = \ell + 2$ while for Timeline PDS parameterization by vimw$[2] + k + \ell$ is not useful. The latter result is obtained by showing NP-hardness of the extremely restricted special case when there are two snapshots, one of which is edgeless, $k = 1$ and $\ell = 0$.

Afterwards, we consider the interval-membership-width (imw). Informally, this is the edge-version of vimw, that is, the bag at timestep i contains the edges which occur at timestep i and those that occur both before and after i. Note that imw can be considered to be a smaller parameter than vimw: For all instances, we have vimw $\in \Omega(\sqrt{\text{imw}})$ and there are instances with constant values of imw and unbounded values of vimw. We show that Timeline DS is fixed-parameter tractable for the parameter imw $+ k + \ell$ while all three other problems are NP-hard for constant values of imw $+ k + \ell$. Given the latter hardness results, we refrain from analyzing a hierarchy of imw-based parameters. Altogether, the results show that in our setting, vimw is a much more powerful parameter than imw.

We then conclude by considering the more standard parameters n, k, ℓ, and t, where t denotes the number of vertices to dominate or edges to cover, respectively. For Timeline DS, we show fixed-parameter tractability for $n + k$ and W[1]-hardness for $n+\ell$, thus showing that the complexity is similar to the one of Timeline VC. Finally, we show that Timeline PDS

Table 1 Overview of our results on the classic and parameterized complexity for Timeline (Partial) Vertex Cover and Timeline (Partial) Dominating Set. Herein, vimw denotes the vertex-interval-membership-width, imw denotes the interval-membership-width, and vimw$[x]$ denotes the size of the x-largest bag in the vertex-interval-membership sequence. Recall that vimw = vimw[1].

Parameter	Timeline VC	Timeline PVC	Timeline DS	Timeline PDS
vimw $+ k + \ell$	FPT Thm. 4.1	FPT Thm. 4.1	FPT Thm. 4.3	FPT Thm. 4.3
imw $+ k + \ell$	NP-h Thm. 4.7	NP-h Thm. 4.7	FPT Thm. 4.6	NP-h Thm. 4.8
$k + \ell +$ vimw$[x]$	$x = k(\ell+1)+1$ FPT Prop. 4.2	$x = k(\ell+1)+1$ FPT Prop. 4.2	$x = \ell + 2$ FPT Prop. 4.4	$x = 2$ NP-h Thm. 4.5
$n + k$	FPT [18, Thm. 8]	?	FPT Thm. 5.1	?
$n + \ell$	W[1]-h [18, Thm. 12]	W[1]-h [18, Thm. 12]	W[1]-h Thm. 5.2	W[1]-h Thm. 5.2
t	–	FPT Thm. 5.4	–	FPT Thm. 5.4

and Timeline PVC can be solved in $2^{\mathcal{O}(t)} \cdot (n+T)^{\mathcal{O}(1)}$ time. This improves and extends a previous result of Dondi et al. [12, Thm. 6] who provided an algorithm for Timeline PVC with $k = 1$ that has running time $2^{t \cdot \log(t)} \cdot (n+T)^{\mathcal{O}(1)} = t^t \cdot (n+T)^{\mathcal{O}(1)}$.

Proofs of statements marked with ($\star$) are deferred to the related full version.

Further related work. Akrida et al. [1] introduced the problems Temporal Vertex Cover (TVC) and Sliding Window Temporal Vertex Cover (SW-TVC), where the goal is to find a minimum number of temporal vertices that cover the edges of the temporal graph. In TVC each edge needs to be covered in at least one of the snapshots and in SW-TVC, the input contains an integer Δ, and the aim is to cover every edge at least once at every Δ consecutive time steps. Akrida et al. [1] showed that TVC remains NP-hard when the underlying graph is a star graph. Moreover, they studied the approximability of both problems, provided (S)ETH-based running time lower bounds and designed an exact dynamic programming algorithm with exponential running time. The research on these problems was then continued by Hamm et al. [20], who showed that TVC is solvable in polynomial time if the underlying graph is a path or cycle while SW-TVC remains NP-hard in this case. Besides that, they provided several (exact and approximation) algorithms for the sliding window variant. The Temporal Dominating Set (TDS) problem is defined analogously, here the task is to dominate every vertex in at least one of the snapshots of the temporal graph by selecting few temporal vertices. TDS is NP-hard in very restricted cases, for example when the maximum degree in every snapshot is 2 [22]. There are further temporal variants of Vertex Cover and Dominating Set which have been discussed and studied. These include a multistage variant of Vertex Cover [17], a reachability-based variant for Dominating Set [23] and different conditions on how and when a vertex should be dominated [4, 31].

2 Preliminaries

We denote the set of integers $\{i, i+1, \ldots, j-1, j\}$ by $[i, j]$, and $[1, i]$ simply by $[i]$. Given a set V and an integer $k > 0$, we write $\binom{V}{k}$ for the collection of all size-k subsets of V.

Graph theory. A *(static) graph* $G = (V, E)$ consists of a set of *vertices* V and a set of edges $E \subseteq \binom{V}{2}$. We denote by $V(G)$ and $E(G)$ the vertex and edge set of G, respectively. Furthermore, we let $n := |V(G)|$ and $m := |E(G)|$. For an edge $\{u, v\}$ we write uv and call u and v *endpoints* of the edge. Further, we say that the vertex $v \in V$ is *incident* with $e \in E$ if $v \in e$. If $uv \in E$, then u and v are *adjacent* and they are *neighbors* of each other. The set $N_G(v) := \{u \in V : u \text{ and } v \text{ are adjacent}\}$ is the *(open) neighborhood* of v. Moreover, $N_G[v] := N_G(v) \cup \{v\}$ is the *closed neighborhood* of v. For a set of vertices S, we let $N_G[S] := \bigcup_{s \in S} N_G[s]$ and $N_G(S) = N_G[S] \setminus S$. By $\deg_G(v) := |N_G(v)|$ we denote the *degree* of v. If $\deg_G(v) = 0$, we say that v is *isolated*. The *maximum degree* is $\Delta(G) := \max_{v \in V} \deg_G(v)$. For vertex sets S and T, we let $E_G(S, T) := \{uv \in E(G) : u \in S \text{ and } v \in T\}$ and we let $E(S) := E(S, S)$. A graph G is *bipartite* if $V(G)$ can be partitioned into two sets S and T such that $E(G) = E(S, T)$. For more details on graph theory, we refer to the book by Diestel [8].

Temporal graphs. A *temporal graph* $\mathcal{G}$ is a finite sequence of graphs $(G_1, \ldots, G_T)$ which all have the same set of vertices $V(\mathcal{G}) := V(G_1) = \cdots = V(G_T)$. The graphs G_i are called *snapshots*, T is the *lifetime of* $\mathcal{G}$, and $i \in [T]$ is a *time step*. We may write V instead of $V(\mathcal{G})$ and E_i instead of $E(G_i)$. Moreover, for $u, v \in V$ and $i \in [T]$, (v, i) is a *temporal vertex* and $(\{u, v\}, i) = (uv, i)$ with $uv \in E_i$ is a *temporal edge* in snapshot G_i. We say edge e *appears* in G_i if $e \in E_i$. Moreover, we say that a vertex v is *incident* with the temporal edge (e, i) if $v \in e$. For a temporal vertex (v, i), we define the *(open) neighborhood* by $N_{\mathcal{G}}((v, i)) := N_{G_i}(v)$ and the *closed neighborhood* by $N_{\mathcal{G}}[(v, i)] := N_{G_i}[v]$. By $G_{\downarrow}(\mathcal{G}) := (V_{\downarrow} := V(\mathcal{G}), E_{\downarrow} := \bigcup_{i=1}^{T} E_i)$ we denote the *underlying graph* of $\mathcal{G}$. We let $\Delta_i(\mathcal{G}) = \Delta(G_i)$. Furthermore, by $\Delta_{\max}(\mathcal{G}) := \max_{i=1}^{T} \Delta_i(\mathcal{G})$ we denote the *maximum snapshot degree* of $\mathcal{G}$. If $E_i = \emptyset$ we say that G_i is *empty*. We may drop the subscript $\cdot_{\mathcal{G}}$ when it is clear from context.

Parameter definitions. We give the precise definitions of the parameters related to the (vertex)-interval-membership width: vimw, vimw$[x]$, and imw.

Let $\mathcal{G} = (G_1, \ldots, G_T)$ be a temporal graph. The *vertex-interval-membership sequence* of $\mathcal{G}$ is the sequence $(F_i)_{i \in [T]}$ of vertex-subsets $F_i \subseteq V(\mathcal{G})$ (called *bags*), where each F_i is defined as

$$F_i := \{v \in V(\mathcal{G}) : \exists p, q \in [T] \text{ such that } p \leq i \leq q, \deg_{G_p}(v) \geq 1 \text{ and } \deg_{G_q}(v) \geq 1\}.$$

The *vertex-interval-membership-width* of $\mathcal{G}$, denoted by vimw$(\mathcal{G})$, is the maximum size of a bag in the vertex-interval-membership sequence, that is, vimw$(\mathcal{G}) := \max\{|F_i| : i \in [T]\}$.

For given $x \in [T]$ we introduce the parameter vimw$[x]$ as the size of the xth largest bag of the vertex-interval-membership sequence. Formally, let $(F_{i_1}, \ldots, F_{i_T})$ be an ordering of $(F_i)_{i \in [T]}$ such that $|F_{i_1}| \geq \cdots \geq |F_{i_T}|$. Then we define

$$\text{vimw}[x](\mathcal{G}) := |F_{i_x}|.$$

Note that vimw[i] can be much smaller than vimw[1] = vimw: for example consider a temporal graph where every vertex only appears non-isolated for a short period of time. If all snapshots have only a few edges, then vimw is small. However, a single snapshot with many edges is enough to make vimw arbitrarily large. If only a few such snapshots exist, then vimw[i] is already much smaller then vimw for small i.

The *interval-membership sequence* of $\mathcal{G}$ is the sequence $(F_i)_{i\in[T]}$ of edge-subsets $F_i \subseteq E_\downarrow$ (again called *bags*), where each F_i is defined as

$$F_i := \{e \in E_\downarrow : \exists p, q \in [T] \text{ such that } p \leq i \leq q \text{ and } e \in E_p \cap E_q\}.$$

The *interval-membership-width* of $\mathcal{G}$, denoted by imw($\mathcal{G}$), is the maximum size of a bag in the interval-membership sequence, that is, imw($\mathcal{G}$) := max$\{|F_i|: i \in [T]\}$.

We remark that vimw $= \Omega(\sqrt{\text{imw}})$ and that there exist temporal graphs with imw $= 1$ but arbitrary large vimw [3]. Finally, note that the vertex-interval-membership sequence and the interval-membership sequence can be computed in polynomial time with respect to the input size.

Timelines. Given a temporal graph $\mathcal{G} = (G_1, \ldots, G_T)$, an *activity interval* of a vertex v is a triple $(v, a, b) \in V(\mathcal{G}) \times [T] \times [T]$ such that $a \leq b$. We say a that is the *starting time* and b is the *end time* of an activity interval (v, a, b). The *length* of the activity interval is defined by $b - a$. A *k-activity timeline* of the temporal graph $\mathcal{G}$ is a set of activity intervals $\mathcal{T} \subseteq \{(v, a, b) \in V(\mathcal{G}) \times [T] \times [T] : a \leq b\}$, which contains at most k activity intervals for each vertex. A *k-activity ℓ-timeline* is a k-activity timeline in which each activity interval has length at most ℓ. Given a k-activity timeline $\mathcal{T}$, a vertex v is called *active* in snapshot G_i, if there exists $(v, a, b) \in \mathcal{T}$ such that $a \leq i \leq b$. A k-activity timeline $\mathcal{T}$ *dominates* the temporal vertex (v, i) if $v \in N_{G_i}[\{u : u \text{ is active in } G_i\}]$. A k-activity timeline $\mathcal{T}$ *covers* the temporal edge (e, i) if at least one of the two endpoints of e is active in snapshot G_i.

Parameterized complexity. Let $L \subseteq \Sigma^*$ be a computational problem specified over some alphabet Σ and let $p : \Sigma^* \to \mathbb{N}$ be a *parameter*, that is, p assigns to each instance of L an integer parameter value (which we simply denote by p if the instance is clear from the context). We say that L is *fixed-parameter tractable (FPT) with respect to p* if L can be decided in $f(p) \cdot |I|^{\mathcal{O}(1)}$ time where $|I|$ is the length of the input. The corresponding hardness concept related to fixed-parameter tractability is W[t]-hardness, $t \geq 1$; if a problem L is W[t]-hard with respect to p, then L is assumed to *not* be fixed-parameter tractable. Moreover, L is *slice-wise polynomial (XP) with respect to p* if L can be decided in $f(p) \cdot |I|^{g(p)}$ time. Let (I, p) and (I', p') be two instances of L. A *reduction to a (problem) kernel for L* is a polynomial-time algorithm that computes an instance (I', p') such that

- $p' + |I'| \leq g(p)$, and
- (I, p) is a yes-instance of L if and only if (I', p') is a yes-instance of L.

The instance (I', p') is referred to as *the problem kernel* and g is the *size* of the kernel. Furthermore, if g is a polynomial, we say that the kernel is a *polynomial problem kernel*. For more details about parameterized complexity we refer to the standard monographs [7, 14].

3 NP-Hardness Results

We show NP-hardness for Timeline VC and Timeline DS even when they are restricted to temporal graphs with a maximum snapshot degree of 1 and constant lifetime T, interval number k, and interval length ℓ. First, we present the reduction for Timeline VC.

▶ **Theorem 3.1.** *Timeline VC is NP-hard even if $T = 23, k = 2, \ell = 4$, and the maximum snapshot degree is one.*

Proof. We reduce from the NP-hard problem 3-Coloring on graphs with maximum degree four [19]. In 3-Coloring the input is a graph and the question is whether the vertices can be colored with three colors such that no two adjacent vertices have the same color.

Intuition. The basic idea is to construct a temporal graph with three parts C_1, C_2 and C_3 (call them *color blocks*), each consisting of five snapshots and representing one of the three colors. The part in which a vertex v is *not active* corresponds to the assigned color of v. As each of the three parts will contain every edge of G and only vertices of degree at most one, this ensures that all neighbors are active in the part where v is not active. Hence, they are assigned a different color. The basic idea for encoding a given graph G into snapshots of maximum degree one is to find a proper edge coloring with five colors and split the edge set with respect to this coloring, which can be done in polynomial time via Vizing's theorem. Our aim is that no activity interval of any vertex can hit more than one color block. To ensure this, we add sufficiently many empty snapshots between any two parts.

Construction. Let $(G = (V = \{v_1, \ldots, v_n\}, E))$ be an instance of 3-Coloring on graphs with maximum degree four. We construct the following temporal graph $\mathcal{G}$ where $V(\mathcal{G}) = V$.

The temporal graph $\mathcal{G}$ consists of 23 snapshots. It can be seen as a sequence of five parts, namely C_1, H_1, C_2, H_2, C_3. Parts H_1 and H_2 consist of 4 empty snapshots and each of C_1, C_2, C_3 consists of 5 snapshots. Thus, snapshots 6 to 9 and 15 to 18 are empty. We denote H_1 and H_2 as the *empty blocks* and we call C_1, C_2 and C_3 *color blocks* as they correspond to the three colors. In the following we formally define the snapshots of the color blocks. All three color blocks C_1, C_2 and C_3 will be identical. Hence, we only formally define C_1, that is, snapshots 1 to 5.

Here we exploit the fact that the graph G has maximum degree four, because this implies that there exists a proper edge coloring with at most five colors due to Vizing's theorem [32], which can be computed in polynomial time [25]. Let $E(G) = F_1 \cup \cdots \cup F_5$ be the partition of $E(G)$ into five colors of some proper edge coloring. Now, snapshot j contains exactly the edges of F_j. Finally, observe that by construction, the maximum degree in each snapshot is at most one. By setting $k = 2$ and $\ell = 4$, we obtain the Timeline VC instance $(\mathcal{G}, k, \ell)$.

Correctness. We show that G is 3-colorable if and only if $\mathcal{G}$ has a 2-activity 4-timeline $\mathcal{T}$ covering all temporal edges of $\mathcal{G}$. In the following we say a vertex v is *active* during C_j for $j \in [3]$ to indicate that an activity interval of v starts at the first time step and ends and the last time step of the corresponding part.

($\Rightarrow$) Let $\phi : V(G) \to [3]$ be a proper 3-coloring of G. We construct an activity timeline of $\mathcal{G}$ which covers all temporal edges. For each $i \in [n]$, the vertex $v \in V(G)$ is active in the two color blocks, which do not correspond to the assigned color of v. Clearly, this yields a 2-activity timeline $\mathcal{T}$. It remains to argue that each temporal edge is covered by $\mathcal{T}$. Since ϕ is a proper 3-coloring, both endpoints of each edge $uw \in E(G)$ have a different color. Moreover, since in color block C_j all vertices are active which do not have color j, we conclude that all temporal edges of $\mathcal{G}$ are covered by $\mathcal{T}$.

($\Leftarrow$) Let $\mathcal{T}$ be a 2-activity 4-timeline covering all temporal edges of $\mathcal{G}$. Observe that since parts H_1 and H_2 corresponding to snapshots 6 to 9 and 15 to 18 are empty, no activity interval of any vertex can hit more than one color block. Consequently, since $k = 2$, for each vertex v there exists at least one color $j \in [3]$ such that v is not active during any snapshot of color block C_j. Let $\phi(v) \in [3]$ be such a color. Note that if $\phi(v)$ is not unique, we pick any color fulfilling the property. We now argue that ϕ is a proper 3-coloring of G.

Assume towards a contradiction that this is not the case, that is, there exists at least one edge $uw \in E(G)$ such that both endpoints u and w have the same color j. This implies that both u and w are not active during C_j. But since exactly one of the 5 snapshots corresponding to color block C_j contains the edge uw, timeline $\mathcal{T}$ is not covering all temporal edges of $\mathcal{G}$, a contradiction. Thus, ϕ is a proper 3-coloring of G. ◀

Next, we extend the ideas of this reduction to Timeline DS for which the reduction is more involved since we also need to deal with isolated temporal vertices.

▶ **Theorem 3.2** (⋆). *Timeline DS is NP-hard even if* $T = 35, k = 3, \ell = 6$ *and the maximum snapshot degree is one.*

4 The Influence of Membership-Width Based Parameters

Now, we investigate the membership-width based parameters vimw and imw. More precisely, we study the parameter combinations $\text{vimw} + k + \ell$ and $\text{imw} + k + \ell$. These combinations are motivated by the fact that Timeline VC and Timeline DS are W[1]-hard with respect to $n + \ell$; for Timeline VC this follows from previous work [18] and for Timeline DS, we will show this in Section 5.

4.1 The Parameter Vertex-Interval-Membership-Width

In this section, we provide FPT-algorithms with respect to $\text{vimw} + k + \ell$ for Timeline PVC and Timeline PDS. Moreover, we show that vimw can be replaced by smaller parameters $\text{vimw}[x]$ for all problems except Timeline PDS (the precise value of x differs for Timeline PVC and Timeline DS). Simultaneously, these algorithms also are XP-algorithms for Timeline PVC and Timeline PDS parameterized by vimw alone. Initially, we focus on Timeline (Partial) Vertex Cover and we show that Timeline PVC admits an FPT-algorithm for $\text{vimw} + k + \ell$.

▶ **Theorem 4.1** (⋆). *Timeline PVC is solvable in* $\big((k+1)(\ell+2)\big)^{2\cdot\text{vimw}} \cdot (n+T)^{\mathcal{O}(1)}$ *time.*

Proof. Let $(\mathcal{G} = (G_1, \ldots, G_T), k, \ell, t)$ be an instance of Timeline PVC. Further, suppose that $T \geq k \cdot (\ell + 1) + 1$ (otherwise, it is a trivial instance) and that the underlying graph $G_\downarrow$ contains no isolated vertex v. Otherwise, since v does not cover any temporal edge, v can be safely removed and consequently t remains unchanged. Moreover, we assume that each vertex v is contained in at least $k \cdot (\ell + 1) + 1$ bags (recall that these are consecutive) because otherwise we can simply greedily pick k activity intervals of v which contain all snapshots where v is non-isolated and thus we cover all temporal edges incident with v. Thus, it is safe to remove v from $\mathcal{G}$ and reduce t by the number of temporal edges incident with v over all snapshots.

The idea is to use dynamic programming over the lifetime of the temporal graph. Each table entry corresponds to the maximal number of covered temporal edges where a specific set of vertices in the bag is active. In other words, the dynamic program keeps track of the activity of the vertices which are contained in the bag F_i of the currently considered time step $i \in [T]$. While iterating over the lifetime of the temporal graph, we need to ensure that the activities at a time step are compatible with the activities at the neighboring time steps. Moreover, for a vertex v let i and j be the index of the smallest and largest snapshot where v is incident to an edge, respectively. By definition, $v \in F_x$ for any $i \leq x \leq j$. Furthermore, since v is isolated in all snapshots G_x where $x < i$ or $x > j$, it is safe to assume that all k activity intervals of v are used when v is contained in the bags F_x.

Recall that the vertex-interval-membership sequence can be computed in polynomial time. Since we assumed that the underlying graph contains no isolated vertex, it follows that each vertex is contained in at least one bag of the sequence.

Table Definition. We denote $\overleftarrow{F}_i := F_1 \cup \cdots \cup F_i$ and $\overline{F}_i := \overleftarrow{F}_i \setminus F_i$. In other words, $\overline{F}_i$ is the set of vertices which are isolated in all snapshots $G_i, \ldots, G_T$ since we assumed that $G_\downarrow$ contains no isolated vertices. Moreover, for $i \in [T]$ we define functions $\mathrm{curr} : F_i \to [0, k]$, and $\mathrm{pos} : F_i \to [0, \ell + 1]$. The functions curr and pos indicate for each vertex $v \in F_i$ whether an activity interval of v is active during snapshot G_i. More precisely, the function value $\mathrm{curr}(v)$ determines the number of the current activity interval of v and the function value $\mathrm{pos}(v)$ determines the position in this activity interval. Here, $\mathrm{curr}(v) = 0$ means that the current time step is before the first activity interval and $\mathrm{pos}(v) = 0$ means that v is currently not active, that is, the $\mathrm{curr}(v)$-th activity interval of v is already over and the next one has not started yet. Now the entry $\mathrm{DP}[i, \mathrm{curr}, \mathrm{pos}]$ contains the maximum number of covered temporal edges in the first i snapshots $G_1, \ldots, G_i$, while i is at the $\mathrm{pos}(v)$-th position of the $\mathrm{curr}(v)$-th activity interval of $v \in F_i$.

The formal definition of the table now is

$$\mathrm{DP}[i, \mathrm{curr}, \mathrm{pos}] \mathrel{\hat{=}} \max_{\mathcal{T}} |\{(e, j) \in E_\downarrow \times [i] : (e, j) \text{ is covered by } \mathcal{T}\}|,$$

where the maximum is taken over all k-activity ℓ-timelines

$$\mathcal{T} \subseteq (\overline{F}_i \times [i-1] \times [T]) \cup (F_i \times [i] \times [T]),$$

such that for each $v \in F_i$

- **(i)** there are at most $\mathrm{curr}(v)$ activity intervals of v in $\mathcal{T}$,
- **(ii)** $\mathrm{pos}(v) \leq i$, and
- **(iii)** $(v, i - \mathrm{pos}(v) + 1, b) \in \mathcal{T}$ for $b = \min(i - \mathrm{pos}(v) + 1 + \ell, T)$ if $\mathrm{pos}(v) > 0$.

We need Conditions (i)-(iii) for the following reasons: Condition (i) ensures that each vertex has at most k activity intervals. Moreover, condition (ii) ensures that no activity interval starts before time step one. Finally, condition (iii) ensures that each activity interval has length ℓ, except if the remaining number of snapshots is too small.

Initialization. For all $\mathrm{curr} : F_1 \to [0, k]$ and $\mathrm{pos} : F_1 \to [0, 1]$, for which $\mathrm{curr}(v) = 0$ implies $\mathrm{pos}(v) = 0$, the table is initialized by

$$\mathrm{DP}[1, \mathrm{curr}, \mathrm{pos}] = |E_{G_1}(\{v \in F_1 : \mathrm{pos}(v) = 1\}, F_1)|.$$

Note that $\overline{F}_1 = \emptyset$ and therefore only activity timelines $\mathcal{T} \subseteq F_1 \times [1] \times [T]$ are considered in the definition of $\mathrm{DP}[1, \mathrm{curr}, \mathrm{pos}]$. The initialization is correct, since the table entry $\mathrm{DP}[1, \mathrm{curr}, \mathrm{pos}]$ contains the number of temporal edges which are covered by the active vertices from F_1 in G_1. Observe that the other endpoints of the covered edges are also contained in F_1 by definition.

Recurrence. The remaining table entries are computed recursively. Intuitively, $\mathrm{DP}[i, \mathrm{curr}, \mathrm{pos}]$ is computed by trying all activity interval choices for the vertices of F_{i-1}, such that there is no conflict with curr and pos. Formally, we set

$$\mathrm{DP}[i, \mathrm{curr}, \mathrm{pos}] = \max_{\mathrm{curr}', \mathrm{pos}'} \mathrm{DP}[i-1, \mathrm{curr}', \mathrm{pos}'] + |E_{G_i}(\{v \in F_i : \mathrm{pos}(v) \geq 1\}, F_i)|$$

where the maximum is taken over all $\mathrm{curr}' : F_{i-1} \to [0, k]$ and $\mathrm{pos}' : F_{i-1} \to [0, \ell + 1]$ which are *compatible* with curr and pos. Informally, curr′ and pos′ are *compatible* with curr and pos if they provide a correct extension of the activities of $F_{i-1} \cap F_i$ at time step i to time

step $i-1$. Formally, four conditions need to be fulfilled: First, if an interval of v is already active at time step i, that is, if $\mathrm{pos}(v) \geq 2$, then an activity interval of v is already active during time step $i-1$. This can be formalized as follows.

$$\text{If } \mathrm{pos}(v) \geq 2, \text{ then } \mathrm{curr}'(v) \leq \mathrm{curr}(v) \text{ and } \mathrm{pos}'(v) = \mathrm{pos}(v) - 1. \tag{1}$$

Second, if a new activity interval of v starts at time step i either v was inactive at time step $i-1$ or the previous activity interval of v ended at time step $i-1$. This can be formalized as follows.

$$\text{If } \mathrm{curr}(v) > 1 \text{ and } \mathrm{pos}(v) = 1, \text{ then } \mathrm{curr}'(v) \leq \mathrm{curr}(v) - 1 \text{ and } \mathrm{pos}'(v) \in \{\min(i-1, \ell+1), 0\}. \tag{2}$$

Third, if the first activity interval of v starts at time step i, then in the previous time step $i-1$ vertex v cannot be active. This can be formalized as follows.

$$\text{If } \mathrm{curr}(v) = 1 \text{ and } \mathrm{pos}(v) = 1, \text{ then } \mathrm{curr}'(v) = 0 \text{ and } \mathrm{pos}'(v) = 0. \tag{3}$$

Finally, if v is not active during time step i, either v is also not active during time step $i-1$ or an activity interval of v ended at time step $i-1$. This can be formalized as follows.

$$\text{If } \mathrm{curr}(v) \geq 1 \text{ and } \mathrm{pos}(v) = 0, \text{ then } \mathrm{curr}'(v) \leq \mathrm{curr}(v) \text{ and } \mathrm{pos}'(v) \in \{\min(i-1, \ell+1), 0\}. \tag{4}$$

The minimum in Conditions 2 and 4 ensures that the activity intervals do not have a starting point smaller than one.

Correctness. We show the correctness of the computation by proving inequalities in both directions.

($\leq$) Suppose the maximum in the definition of the table entry $\mathrm{DP}[i, \mathrm{curr}, \mathrm{pos}]$ is attained for $\mathcal{T}$ and assume that no activity intervals from $\mathcal{T}$ overlap. We define $\mathcal{T}'$ by taking all activity intervals from $\mathcal{T}$ which are relevant for a table entry of $i-1$. Formally, we define

$$\begin{aligned}\mathcal{T}' :=& \{(v,a,b) \in \mathcal{T} : a \leq i-2, v \in \overline{F}_{i-1}\} \\ &\cup \{(v,a,b) \in \mathcal{T} : a \leq i-1, v \in F_{i-1}\},\end{aligned}$$

and, in accordance with $\mathcal{T}'$, functions $\mathrm{curr}' : F_{i-1} \to [0,k], \mathrm{pos}' : F_{i-1} \to [0, \ell+1]$ by

$$\mathrm{curr}'(v) := |\{(v,a,b) \in \mathcal{T}' : a, b \in [T]\}|$$

$$\mathrm{pos}'(v) := \begin{cases} i-a & \text{if } (v,a,b) \in \mathcal{T}' \text{ with } a \leq i \leq b \\ 0 & \text{otherwise.} \end{cases}$$

Note that $\mathcal{T}' \subseteq \mathcal{T}$. We show that curr' and pos' satisfy Conditions (1)–(4). This means that the table entry $\mathrm{DP}[i-1, \mathrm{curr}', \mathrm{pos}']$ is considered in the maximum on the right side of the recursive computation. By definition of curr' and pos' it is clear that $\mathcal{T}'$ is then considered in the definition of the table entry $\mathrm{DP}[i-1, \mathrm{curr}', \mathrm{pos}']$. Now let $v \in F_{i-1} \cap F_i$.

(1) If $\mathrm{pos}(v) \geq 2$, then $(v,a,b) \in \mathcal{T}$ for some $a \leq i-1, b \in [T]$ and by definition of $\mathcal{T}'$, each such activity interval of v in $\mathcal{T}$ is also contained in $\mathcal{T}'$. Consequently, we have $\mathrm{curr}'(v) \leq \mathrm{curr}(v)$ and $\mathrm{pos}'(v) = \mathrm{pos}(v) - 1$.

(2) If $\mathrm{curr}(v) > 1$ and $\mathrm{pos}(v) = 1$, then $(v,i,b) \in \mathcal{T} \setminus \mathcal{T}'$ for some $b \in [T]$. So the activity timeline $\mathcal{T}'$ contains at most $\mathrm{curr}(v) - 1$ activity intervals of v and therefore $\mathrm{curr}'(v) \leq \mathrm{curr}(v) - 1$. Now either $(v,a,i-1) \in \mathcal{T}'$ for some $a \in [i-1]$ or v is not active in G_{i-1} with respect to the activity intervals from $\mathcal{T}'$. This means that either $\mathrm{pos}'(v) = \min(i-1, \ell+1)$ or $\mathrm{pos}'(v) = 0$.

(3) If $\mathrm{curr}(v) = 1$ and $\mathrm{pos}(v) = 1$, then $(v, i, b) \in \mathcal{T} \setminus \mathcal{T}'$ for some $b \in [T]$. Consequently, $\mathcal{T}'$ contains no activity interval of v and therefore $\mathrm{curr}'(v) = \mathrm{pos}'(v) = 0$.

(4) If $\mathrm{curr}(v) \geq 1$ and $\mathrm{pos}(v) = 0$, then either $(v, a, i-1) \in \mathcal{T}'$ for some $a \in [i-1]$ or v is not active in G_{i-1} with respect to the activity intervals from $\mathcal{T}'$. So by definition $\mathrm{curr}'(v) \leq \mathrm{curr}(v)$ and either $\mathrm{pos}'(v) = \min(i-1, \ell+1)$ or $\mathrm{pos}'(v) = 0$.

Now consider the set $\mathcal{A}$ of temporal edges until time step i that are covered by $\mathcal{T}$ and the set $\mathcal{A}'$ of temporal edges until time step $i-1$ that are covered by $\mathcal{T}'$. There are two cases for a temporal edge $(vw, j) \in \mathcal{A} \setminus \mathcal{A}'$.

Case 1: $j = i$. Since the temporal edge (vw, j) is covered by $\mathcal{T}$ we can conclude that $vw \in E_{G_i}(\{v \in F_i : \mathrm{pos}(v) \geq 1\}, F_i)$.

Case 2: $j < i$. We have $(vw, j) \in \binom{\overleftarrow{F}_{i-1}}{2} \times [i-1]$. This implies the existence of an activity interval (v, a, b) or (w, a, b) in $\mathcal{T}$ with $a \leq i-1$. However, then the activity interval is by definition contained in $\mathcal{T}'$ and consequently $(vw, j) \in \mathcal{A}'$.

Hence, any temporal edge in $\mathcal{A} \setminus \mathcal{A}'$ is contained in $E_{G_i}(\{v \in F_i : \mathrm{pos}(v) \geq 1\}, F_i)$. We thus have the desired inequality

$$\begin{aligned}\mathrm{DP}[i, \mathrm{curr}, \mathrm{pos}] = |\mathcal{A}| &\leq |\mathcal{A}'| + |E_{G_i}(\{v \in F_i : \mathrm{pos}(v) \geq 1\}, F_i)| \\ &\leq \mathrm{DP}[i-1, \mathrm{curr}', \mathrm{pos}'] + |E_{G_i}(\{v \in F_i : \mathrm{pos}(v) \geq 1\}, F_i)|.\end{aligned}$$

($\geq$) Suppose the maximum on the right side of the computation is attained for $\mathrm{curr}', \mathrm{pos}'$ and the maximum in the definition of $\mathrm{DP}[i-1, \mathrm{curr}', \mathrm{pos}']$ is attained for $\mathcal{T}'$. We define $\mathcal{T}$ by extending $\mathcal{T}'$ in the following way: For $v \in F_{i-1} \cap F_i$ the activity interval $(v, i, \min(i+\ell, T))$ is added if and only if $\mathrm{pos}(v) = 1$. These new intervals together with the already active ones ($\mathrm{pos}(v) \geq 2$) cover all edges of $E_{G_i}(\{v \in F_i : \mathrm{pos}(v) \geq 1\}, F_i)$. Hence, the total number of edges covered by $\mathcal{T}$ is $\mathrm{DP}[i-1, \mathrm{curr}', \mathrm{pos}'] + |E_{G_i}(\{v \in F_i : \mathrm{pos}(v) \geq 1\}, F_i)|$.

Because $\mathcal{T}$ is considered in the definition of $\mathrm{DP}[i, \mathrm{curr}, \mathrm{pos}]$, we have

$$\mathrm{DP}[i, \mathrm{curr}, \mathrm{pos}] \geq \mathrm{DP}[i-1, \mathrm{curr}', \mathrm{pos}'] + |E_{G_i}(\{v \in F_i : \mathrm{pos}(v) \geq 1\}, F_i)|.$$

Hence, the desired inequality holds.

Result. Finally, we return yes if and only if $\mathrm{DP}[T, \mathrm{curr}, \mathrm{pos}] \geq t$ for $\mathrm{curr} : F_T \to \{k\}$ and some function $\mathrm{pos} : F_T \to \{0, \ell+1\}$. The restriction to these functions is correct, since we can assume without loss of generality that a vertex is active in G_T if and only if its k-th activity interval ends at time step T (recall that we assume that each vertex is in at least $k(\ell+1)+1$ bags). Consequently, all k-activity ℓ-timelines of $\mathcal{G}$ are considered in the definition of the table entry $\mathrm{DP}[T, \mathrm{curr}, \mathrm{pos}]$ and all covered temporal edges of $\mathcal{G}$ are counted in the respective maximum of the definition.

Running Time. For each time step $i \in [T]$, there are $\big((k+1)(\ell+2)\big)^{\mathrm{vimw}}$ choices for the functions $\mathrm{curr} : F_i \to [0, k]$ and $\mathrm{pos} : F_i \to [0, \ell+1]$. This upper-bounds the size of our dynamic programming table by $T \cdot \big((k+1)(\ell+2)\big)^{\mathrm{vimw}}$. In each recursive computation of a table entry, there are also $\mathcal{O}(\big((k+1)(\ell+2)\big)^{\mathrm{vimw}})$ choices for curr' and pos'. The remaining summands of the recursive formula can be computed in $\mathcal{O}(n)$ time. Hence, the overall running time can be bounded by $\big((k+1)(\ell+2)\big)^{2 \cdot \mathrm{vimw}} \cdot (n+T)^{\mathcal{O}(1)}$. ◀

Now, we show that for Timeline PVC the parameter $\mathrm{vimw} = \mathrm{vimw}[1]$ can be replaced by an even smaller parameter, $\mathrm{vimw}[k(\ell+1)+1]$. The algorithm for this parameter has two steps: First it applies a preprocessing step that handles the large bags. Then, it invokes the algorithm of Theorem 4.1 for $\mathrm{vimw} = \mathrm{vimw}[1]$.

▶ **Proposition 4.2.** *Timeline PVC is solvable in* $\big((k+1)(\ell+2)\big)^{4\cdot\mathrm{vimw}[x]} \cdot (n+T)^{\mathcal{O}(1)}$ *time where* $x = k(\ell+1)+1$.

Proof. Let $(F_i)_{i\in[T]}$ be the bags of the vertex-interval-membership sequence of $\mathcal{G}$. Similar to the proof of Theorem 4.1, it is safe to assume that each vertex v is contained in at least $k\cdot(\ell+1)$ bags (by definition the bags containing v need to be consecutive) because otherwise, we can greedily pick the k-activity intervals of v such that v is active during all these snapshots and thus all temporal edges incident to v are covered and we can reduce t accordingly.

We call the $k\cdot(\ell+1)$ largest bags of $(F_i)_{i\in[T]}$ *large*. The idea is to pick the intervals greedily for all vertices that are only contained in large bags. Intuitively, this works since the large bags appear consecutively. Afterwards, we remove all these vertices and adapt t accordingly. Finally, we can invoke the algorithm of Theorem 4.1.

Now, let $[i,j]\subset[T]$ be a sequence of consecutive indices such that each bag F_x for each $x\in[i,j]$ is large. By definition $j-i\leq k\cdot(\ell+1)-1$. Now, let $\mathcal{A}=\bigcup_{x\in[i,j]}F_x\setminus(F_{i-1}\cup F_{j+1})$. If $i=0$ or $j=T$, then F_{i-1} or F_{j+1} is simply the empty set. Suppose $\mathcal{A}\neq\emptyset$. By definition, each vertex in $\mathcal{A}$ is isolated in $G_1,\ldots,G_{i-1}$ and also in $G_{j+1},\ldots,G_T$. By our assumption that each vertex is contained in at least $k\cdot(\ell+1)$ bags, we can conclude that $j-i=k\cdot(\ell+1)-1$. Now observe that by picking the k-activity intervals for each vertex in $\mathcal{A}$ greedily, we can cover all temporal edges which are incident to some vertex of $\mathcal{A}$. Hence, it is safe to remove $\mathcal{A}$ from $\mathcal{G}$ and to reduce t by the number of temporal edges covered by the vertices in $\mathcal{A}$. For the remaining instance, we use the algorithm of Theorem 4.1. After removing $\mathcal{A}$, we have $|F_x|\leq|F_{i-1}|+|F_{j+1}|\leq 2\,\mathrm{vimw}[k(\ell+1)+1]$ for every $x\in[i,j]$ and consequently Theorem 4.1 yields the desired running time. Now, suppose $\mathcal{A}=\emptyset$ and let F_x be a large bag contained in a sequence of large bags with indices $[i,j]$. In this case, we can conclude $|F_x|\leq|F_{i-1}|+|F_{j+1}|\leq 2\,\mathrm{vimw}[k(\ell+1)+1]$ and again apply Theorem 4.1. ◀

Now, we focus on Timeline (Partial) Dominating Set. Initially, we show how to adapt the algorithm of Theorem 4.1 for Timeline PDS to obtain an algorithm with the same running time for Timeline PDS. However, we need to be more careful since isolated vertices in the snapshots can no longer be ignored as for Timeline PVC. This means that vertices which are not contained in the current bag of the vertex-interval-membership sequence might have to be active. Hence, the update of the dynamic program additionally needs to consider the case that activity intervals of vertices appear before or after their first or last occurrence in a bag.

▶ **Theorem 4.3** (⋆). *Timeline PDS is solvable in* $\big((k+1)(\ell+2)\big)^{2\cdot\mathrm{vimw}}\cdot(n+T)^{\mathcal{O}(1)}$ *time.*

Recall that for Timeline PVC we showed that $\mathrm{vimw}=\mathrm{vimw}[1]$ can be replaced by the much smaller parameter $\mathrm{vimw}[k\cdot(\ell+1)+1]$, the size of the $k\cdot(\ell+1)+1$ largest bag. Hence, one may ask whether vimw can be replaced by $\mathrm{vimw}[i]$ with some suitably chosen i for Timeline DS and Timeline PDS. We show that both problems behave quite differently: First, we show we that vimw can be replaced by $\mathrm{vimw}[\ell+2]$ for Timeline DS by having a preprocessing step and then adapting the algorithm behind Theorem 4.3. Second, we show that replacing $\mathrm{vimw}=\mathrm{vimw}[1]$ by $\mathrm{vimw}[2]$ is not possible for Timeline PDS. More precisely, we show NP-hardness even if there are only two snapshots one of which is edgeless (which implies $\mathrm{vimw}[2]=0$).

We start with our result for Timeline DS.

▶ **Proposition 4.4** (⋆). *Timeline DS is solvable in* $\big((k+1)(\ell+2)\big)^{4\cdot\mathrm{vimw}[x]}\cdot(n+T)^{\mathcal{O}(1)}$ *time where* $x=\ell+2$.

Finally, we show hardness for constant k, ℓ, and vimw[2] for Timeline PDS.

▶ **Theorem 4.5** (⋆). *Timeline PDS is NP-hard even if $T = 2$, $k = 1$, $\ell = 0$, the underlying graph is bipartite, planar, has maximum degree 3, and one snapshot is edgeless.*

4.2 The Parameter Interval-Membership-Width

Now, we study the influence of imw instead of vimw. Initially, we show that the parameter $\text{imw} + k + \ell$ yields a polynomial kernel for Timeline DS. This also implies a polynomial kernel for $\text{vimw} + k + \ell$ and $n + k + \ell$ for Timeline DS. More precisely, we show an even stronger result: We provide a polynomial kernel for $q + k + \ell$, where q is the maximal number of edges in any snapshot. Note that q is never larger than imw, since any bag F_i contains E_i. Moreover, in the temporal graph $\mathcal{G}$ with a star on n leaves as underlying graph and lifetime $T = 2n$, where the edge towards the i-th leaf is active at snapshots i and $i + n$, we have $\text{imw} = n$ and $q = 1$.

▶ **Theorem 4.6.** *Timeline DS has a kernel where both the number of snapshots and vertices are cubic in $q + k + \ell$ which can be computed in linear time with respect to the input size.*

Proof. In order to provide the kernel we bound the number of snapshots T and the number n of vertices of the underlying graph based on the following case distinction.

Case 1: $q \geq n/2$. By definition each k-activity ℓ-timeline has active vertices in at most $n \cdot k \cdot (\ell + 1)$ snapshots. Hence, if $T > 2 \cdot q \cdot k \cdot (\ell + 1) \geq n \cdot k \cdot (\ell + 1)$, then the instance is a no-instance.

Case 2.1: $q < n/2$ and $T > 2 \cdot k \cdot (\ell + 1) + 1$. We show that in this case, we are dealing with a no-instance. Since each snapshot contains at most q edges, any set of p vertices can dominate at most $p + q$ vertices in this snapshot. As $q < n/2$, it follows that at least $n/2$ vertices must be active in order to dominate all temporal vertices of the corresponding snapshot G_x. Moreover, note that the total number of active vertices in any k-activity ℓ-timeline is at most $z = n \cdot k \cdot (\ell+1)$. Consequently, $T \leq z/(n/2) \leq 2 \cdot k \cdot (\ell+1)$ for any yes-instance.

Case 2.2: $q < n/2$ and $T \leq 2 \cdot k \cdot (\ell + 1) + 1$. If $n \leq 4 \cdot q \cdot k \cdot (\ell + 1)$, then we have the desired bound. Otherwise, if $n \geq 4 \cdot q \cdot k \cdot (\ell+1)+1$, we can solve the instance in polynomial time as follows. Since each snapshot has at most q edges, at most $2q$ vertices are non-isolated in each snapshot. Moreover, since there are at most $2 \cdot k \cdot (\ell + 1)$ snapshots, in total at most $4 \cdot q \cdot k \cdot (\ell+1)$ vertices are non-isolated in the temporal graph. Consequently, since $n \geq 4 \cdot q \cdot k \cdot (\ell + 1) + 1$ there exists at least one vertex v which is isolated in every snapshot. Thus, if $T \geq k \cdot (\ell+1)+1$ we deal with a trivial no-instance and if $T \leq k \cdot (\ell+1)$ we deal with a trivial yes-instance.

Thus, in each case we either solve the instance (giving a kernel of constant size) or obtain a kernel with at most $2 \cdot q \cdot k \cdot (\ell+1)$ snapshots and at most $4 \cdot q \cdot k \cdot (\ell + 1)$ vertices. Moreover, the kernel can be computed in linear time as it involves only computing n, q, and T. ◀

We show that the remaining problems are NP-hard if $\text{imw} + k + \ell \in \mathcal{O}(1)$.

▶ **Theorem 4.7** (⋆). *Timeline VC is NP-hard even if $k = 2$, $\ell = 0$, $\text{imw} = 4$, each edge of the underlying graph exists in exactly one snapshot, and the underlying graph has a constant maximum degree.*

▶ **Theorem 4.8** (⋆). *Timeline PDS is NP-hard even if $k = 1$, $\ell = 0$, $\text{imw} = 36$ and the underlying graph has a constant maximum degree.*

5 Complexity with respect to Input Parameters

Finally, we study the influence of the input parameters n, k, ℓ, and t on the complexity of Timeline (Partial) Vertex Cover and Timeline (Partial) Dominating Set. Both Timeline VC and Timeline DS are NP-hard for constant values of k and ℓ. Consequently, larger parameters need to be considered to obtain FPT-algorithms or XP-algorithms. Froese et al. [18, Theorem 8] showed that Timeline VC admits an FPT-algorithm for $n + k$. A similar algorithm also works for Timeline DS.

▶ **Theorem 5.1** (⋆). *Timeline DS is solvable in time $\mathcal{O}(n^{2+nk}T)$.*

Froese et al. [18, Theorem 12] showed that Timeline VC is W[1]-hard for n even if $\ell = 1$ with a reduction from Unary Bin Packing and using a multicolored and a nonuniform variant of Timeline VC as intermediate problems in the reduction. In Nonuniform Timeline VC we are not given a single number of permitted activity intervals k, but instead a number k_v for each vertex v. In a similar way we introduce Nonuniform Timeline DS, which we will use as an intermediate problem to show hardness for $n + \ell$ for Timeline DS.

▶ **Theorem 5.2** (⋆). *Timeline DS parameterized by n is* W[1]*-hard even if $\ell = 1$.*

The W[1]-hardness results for both problems do not apply to the case $\ell = 0$. For Timeline VC, this case is fixed-parameter tractable because there exists an ILP formulation in which the number of variables is bounded by some function of n [18, Lemma 10]. It is also straightforward to extend this ILP formulation to Timeline PVC. To complete the picture, we extend the ILP formulation also to Timeline PDS, thus showing that it is fixed-parameter tractable for $\ell = 0$ as well.

▶ **Theorem 5.3** (⋆). *Timeline PDS is fixed-parameter tractable with respect to n if $\ell = 0$.*

Finally, we show that both Timeline PDS and Timeline PVC can be solved in $2^{\mathcal{O}(t)} \cdot (n+T)^{\mathcal{O}(1)}$ time, where t denotes the number of temporal vertices/edges, which need to be dominated/covered. Recall that Dondi et al. [12, Theorem 6] provided an FPT-algorithm for Timeline PVC with running time $t^t \cdot (n+T)^{\mathcal{O}(1)}$ if $k = 1$. Hence, our approach improves upon their running time and is not limited to $k = 1$. The idea is to use color coding [2], a very popular technique to obtain FPT-algorithms [7]. To provide deterministic algorithm, we use (n, k)*-perfect hash families* which can be computed efficiently [26].

▶ **Theorem 5.4** (⋆). *Timeline PDS and Timeline PVC are solvable in $2^{\mathcal{O}(t)} \cdot (n + T)^{\mathcal{O}(1)}$ time.*

6 Conclusion

We studied the classical and parameterized complexity of Timeline (Partial) Vertex Cover and Timeline (Partial) Dominating Set. We showed that all problems admit FPT-algorithms for $\mathrm{vimw} + k + \ell$, where vimw is the vertex-interval-membership width. Our running time bounds also give XP-algorithms for vimw alone. For Timeline VC this improves upon an XP-algorithm for n [18]. Moreover, we showed that the smaller parameter $\mathrm{imw} + k + \ell$ leads to para-NP-hardness for all problems except Timeline DS. Hence, we discovered a parameter where a dominating set problem is tractable while the corresponding vertex cover variant is not.

For future work it is interesting to investigate whether already the parameter vimw $+ k$ yields an FPT-algorithm for any problem in our study. Moreover, it is open whether Timeline PVC and Timeline PDS admit an FPT-algorithm with respect to $n + k$. Even for $k = 1$ fixed-parameter tractability with respect to n is still open. Moreover, one could investigate the complexity with respect to other temporal graph parameters such as the temporal neighborhood diversity [15] or temporal graph parameters that are sensitive to the order of the snapshots [5, 6].

A further problem related to Timeline VC is MinTimeline$_+$. In this problem, the value of ℓ bounds the sum of the lengths of the activity intervals instead of the length of the longest activity interval. MinTimeline$_+$ was also introduced by Rozhenstein et al. [28] and studied in several works [9–11, 13, 18, 24, 29]. To distinguish this problem from the other problems in our naming convention, Timeline VC (MinTimeline$_\infty$) could be called MinMax Timeline VC and MinTimeline$_+$ could be called Sum Timeline VC, and the timeline variants of Dominating Set could be named analogously. It is open which of our positive results for vimw and imw can be transferred to the Sum variants of the problems. Finally, it is interesting to study other classic problems as Feedback Vertex Set in the MinMax Timeline and Sum Timeline setting.

References

1 Eleni C. Akrida, George B. Mertzios, Paul G. Spirakis, and Viktor Zamaraev. Temporal vertex cover with a sliding time window. *J. Comput. Syst. Sci.*, 107:108–123, 2020. doi:10.1016/J.JCSS.2019.08.002.

2 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995. doi:10.1145/210332.210337.

3 Benjamin Merlin Bumpus and Kitty Meeks. Edge exploration of temporal graphs. *Algorithmica*, 85(3):688–716, 2023. doi:10.1007/S00453-022-01018-7.

4 Arnaud Casteigts. *A Journey through Dynamic Networks (with Excursions)*. Habilitation thesis, Université de Bordeaux, 2018. URL: https://tel.archives-ouvertes.fr/tel-01883384.

5 Arnaud Casteigts, Nils Morawietz, and Petra Wolf. Distance to Transitivity: New Parameters for Taming Reachability in Temporal Graphs. In *Proceedings of the 49th International Symposium on Mathematical Foundations of Computer Science (MFCS '24)*, volume 306 of *LIPIcs*, pages 36:1–36:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.MFCS.2024.36.

6 Filippos Christodoulou, Pierluigi Crescenzi, Andrea Marino, Ana Silva, and Dimitrios M. Thilikos. Making the interval membership width of temporal graphs connected and bidirectional. In *Proceedings of the 35th International Workshop on Combinatorial Algorithms (IWOCA '24)*, volume 14764 of *Lecture Notes in Computer Science*, pages 247–258. Springer, 2024. doi:10.1007/978-3-031-63021-7_19.

7 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

8 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate Texts in Mathematics*. Springer, 2012.

9 Riccardo Dondi. Insights into the Complexity of Disentangling Temporal Graphs. In *Proceedings of the 23rd Italian Conference on Theoretical Computer Science (ICTCS '22)*, volume 3284 of *CEUR Workshop Proceedings*, pages 1–13. CEUR-WS.org, 2022. URL: https://ceur-ws.org/Vol-3284/2973.pdf.

10 Riccardo Dondi. Untangling temporal graphs of bounded degree. *Theor. Comput. Sci.*, 969:114040, 2023. doi:10.1016/J.TCS.2023.114040.

11 Riccardo Dondi and Manuel Lafond. An FPT algorithm for timeline cover. *J. Comput. Syst. Sci.*, 154:103679, 2025. doi:10.1016/J.JCSS.2025.103679.

12 Riccardo Dondi, Fabrizio Montecchiani, Giacomo Ortali, Tommaso Piselli, and Alessandra Tappini. Partial temporal vertex cover with bounded activity intervals. In *Proceedings of the 3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND '24)*, volume 292 of *LIPIcs*, pages 11:1–11:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.SAND.2024.11.

13 Riccardo Dondi and Alexandru Popa. Exact and approximation algorithms for covering timeline in temporal graphs. *Ann. Oper. Res.*, 351(1):609–628, 2025. doi:10.1007/s10479-024-05993-8.

14 Rodney G. Downey and Michael Ralph Fellows. *Fundamentals of Parameterized Complexity*. Springer Science & Business Media, 2013.

15 Jessica A. Enright, Samuel D. Hand, Laura Larios-Jones, and Kitty Meeks. Structural parameters for dense temporal graphs. In *Proceedings of the 49th International Symposium on Mathematical Foundations of Computer Science (MFCS '24)*, volume 306 of *LIPIcs*, pages 52:1–52:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.MFCS.2024.52.

16 David Eppstein and Emma S. Spiro. The h-index of a graph and its application to dynamic subgraph statistics. *J. Graph Algorithms Appl.*, 16(2):543–567, 2012. doi:10.7155/JGAA.00273.

17 Till Fluschnik, Rolf Niedermeier, Valentin Rohm, and Philipp Zschoche. Multistage vertex cover. *Theory Comput. Syst.*, 66(2):454–483, 2022. doi:10.1007/s00224-022-10069-w.

18 Vincent Froese, Pascal Kunz, and Philipp Zschoche. Disentangling the computational complexity of network untangling. *Theory Comput. Syst.*, 68(1):103–121, 2024. doi:10.1007/S00224-023-10150-Y.

19 M. R. Garey, David S. Johnson, and Larry J. Stockmeyer. Some simplified NP-complete problems. In *Proceedings of the 6th Annual ACM Symposium on Theory of Computing*, pages 47–63. ACM, 1974. doi:10.1145/800119.803884.

20 Thekla Hamm, Nina Klobas, George B. Mertzios, and Paul G. Spirakis. The complexity of temporal vertex cover in small-degree graphs. In *Proceedings of the Thirty-Sixth AAAI Conference on Artificial Intelligence (AAAI '22)*, pages 10193–10201. AAAI Press, 2022. doi:10.1609/AAAI.V36I9.21259.

21 Anton Herrmann. On the complexity of computing optimal dominating sets in temporal graphs. Master's thesis, Friedrich-Schiller-Universität Jena, 2024.

22 Anton Herrmann, Christian Komusiewicz, Nils Morawietz, and Frank Sommer. Temporal dominating set and temporal vertex cover under the lense of degree restrictions. In *Proceedings of the 4th Symposium on Algorithmic Foundations of Dynamic Networks (SAND '25)*, volume 330 of *LIPIcs*, pages 16:1–16:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPICS.SAND.2025.16.

23 David C. Kutner and Laura Larios-Jones. Temporal reachability dominating sets: Contagion in temporal graphs. *J. Comput. Syst. Sci.*, 155:103701, 2026. doi:10.1016/J.JCSS.2025.103701.

24 Giorgio Lazzarinetti, Sara Manzoni, Italo Zoppis, and Riccardo Dondi. FastMinTC+: A Fast and Effective Heuristic for Minimum Timeline Cover on Temporal Networks. In *Proceedings of the 31st International Symposium on Temporal Representation and Reasoning (TIME '24)*, volume 318 of *LIPIcs*, pages 20:1–20:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.TIME.2024.20.

25 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.

26 Moni Naor, Leonard J. Schulman, and Aravind Srinivasan. Splitters and near-optimal derandomization. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS '95)*, pages 182–191. IEEE Computer Society, 1995. doi:10.1109/SFCS.1995.492475.

27 Tayler Pino, Salimur Choudhury, and Fadi Al-Turjman. Dominating set algorithms for wireless sensor networks survivability. *IEEE Access*, 6:17527–17532, 2018. doi:10.1109/ACCESS.2018.2819083.

28 Polina Rozenshtein, Nikolaj Tatti, and Aristides Gionis. The network-untangling problem: from interactions to activity timelines. *Data Min. Knowl. Discov.*, 35(1):213–247, 2021. doi:10.1007/S10618-020-00717-5.

29 Carsten Schubert. Leveraging graph structure to untangle temporal networks efficiently. Master's thesis, TU Berlin, Institute of Software Engineering and Theoretical Computer Science, 2023.

30 I. Stojmenovic, M. Seddigh, and J. Zunic. Dominating sets and neighbor elimination-based broadcasting algorithms in wireless networks. *IEEE Trans. Parallel Distributed Syst.*, 13(1):14–25, 2002. doi:10.1109/71.980024.

31 M Verheije. Algorithms for domination problems on temporal graphs. Master's thesis, Utrecht University, Graduate School of Natural Sciences, 2021.

32 Vadim G Vizing. On an estimate of the chromatic class of a p-graph. *Discret Analiz*, 3:25–30, 1964.

On Maximum 2-Clubs

Joanne Dumont
Université d'Orléans, INSA CVL, LIFO, UR 4022, Orléans, France

Michael Lampis
Université Paris-Dauphine, PSL University, CNRS UMR7243, LAMSADE, Paris, France

Mathieu Liedloff
Université d'Orléans, INSA CVL, LIFO, UR 4022, Orléans, France

Anthony Perez
Université d'Orléans, INSA CVL, LIFO, UR 4022, Orléans, France

Ioan Todinca
Université d'Orléans, INSA CVL, LIFO, UR 4022, Orléans, France

Abstract

We consider the Maximum 2-Club problem where one is given as input an undirected graph $G = (V, E)$ and seeks a subset of vertices S of maximum size such that any pair of vertices in S is connected by a path of length at most 2 in the graph induced by S. This problem is a natural relaxation of the famous Maximum Clique problem where any pair of vertices must be connected by an edge. Maximum 2-Club has been well-studied and is known to be NP-complete even on split graphs. It can be solved exactly in $O^*(1.62^n)$ time, where n denotes the number of vertices of the input graph, while being polynomial-time solvable on several graph classes. Parameterized algorithms for structural parameters have also been considered, leading in particular to an algorithm with a double-exponential dependence in the parameter *treewidth*. Such an algorithm is actually the best one known for the larger parameter *vertex cover size* up to a constant in the exponent. We provide new results in both directions. We first prove that the double-exponential dependence for parameter vertex cover size is unavoidable under the Exponential Time Hypothesis (ETH). This answers a question left open by Hartung, Komusiewicz, Nichterlein and Suchý [14]. Our result also implies that the problem cannot be solved in time sub-exponential in n even for split graphs. We then provide an exact algorithm for the problem restricted to chordal graphs, running in $O^*(1.1996^n)$ time, by reducing Maximum 2-Club on this class to Maximum Independent Set on arbitrary graphs with the same number of vertices. The same reduction shows that we can enumerate all maximum (and inclusion-wise maximal) 2-clubs of a chordal graph in $O^*(3^{\frac{n}{3}}) = O^*(1.4423^n)$ time. We conclude by providing a construction of split graphs with $\Omega(3^{\frac{n}{3}} / \operatorname{poly}(n))$ maximum 2-clubs, for some polynomial poly showing that the bound for enumeration is essentially tight.

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

Keywords and phrases 2-clubs, chordal graphs, SETH, parameterized algorithms

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.13

Acknowledgements We would like to thank the anonymous referees for providing insightful comments and references which helped improve the presentation of the paper.

1 Introduction

Finding large sets of densely connected subgraphs in a given graph has been a challenging task for decades, with applications ranging from social networks to biological networks (see, e.g., [10]). To that extent, one of the most famous problems is Maximum Clique where one seeks the largest subset of vertices pairwise connected by an edge. Maximum Clique is one of the earliest NP-hard problems [9] and, from a parameterized complexity viewpoint, one of the most classical examples of a W[1]-hard problem (see e.g. [7]). However, the problem

© Joanne Dumont, Michael Lampis, Mathieu Liedloff, Anthony Perez, and Ioan Todinca;
licensed under Creative Commons License CC-BY 4.0

becomes tractable on many well-studied graph classes such as perfect graphs, circle graphs or unit disk graphs, see e.g. [5, 12, 22]. In particular, it becomes easy on many classes of *sparse graphs*, that often contain polynomially many maximal cliques [19, 21, 25]. Nonetheless, such graphs generally do not contain large cliques which is one of the reasons for considering relaxations of the notion of clique.

Many properties of being a clique can be used to define such relaxations. For instance, considering the degree leads to the notion of *p-core* where every vertex in the set S must have degree at least p in the subgraph induced by S. A closely related notion is that of a *p-plex*, where each vertex of S is required to have degree at least $|S| - p$. When considering density, namely the fraction of present edges with respect to the total possible number, the natural relaxation is that of a δ-quasi-clique where the density must be at least δ (note that $\delta = 1$ amounts to a clique while $\delta = 0$ can be any subset of the vertices). We refer the reader to the paper of Komusiewicz [16] and the thesis of Castillon [3] for a more comprehensive description of such relaxations.

In this work, we consider relaxations that are connected to the diameter of the sought subset. The diameter of a graph is the length of its longest shortest path. Two relaxations can be naturally defined with this property: for a given integer s, one may either seek for the largest set S where the distance between any two vertices is at most s *in* G or require that this property holds in the subgraph of G *induced by* S. The former case leads to the notion of *s-clique* while the latter is called *s-club*. We focus on the notion of s-club and especially the case $s = 2$ which corresponds to the notion closest to being a clique. Formally, we consider the following problem.

Maximum 2-club

Input: A simple, undirected graph $G = (V, E)$
Output: Does there exist a 2-club of size at least ℓ in G?

Related work

This problem is well-studied from both the structural and algorithmic viewpoints. As for the Maximum Clique problem, the largest 2-club can be found in polynomial time in many graph classes such as bipartite graphs, trees and interval graphs [23]. Similarly, Golovach et al. [11] prove the polynomial-time solvability of the s-club problem on chordal bipartite, strongly chordal and distance-hereditary graphs. In sharp contrast, Maximum s-Club is NP-complete on graphs of diameter $s + 1$ for all $s \geqslant 1$. The Maximum 2-Club problem is NP-complete even on split graphs and, thus, also on chordal graphs [1]. Moreover, the complexity of s-club is well-understood in a large class of graphs, called weakly chordal graphs: it is polynomial-time solvable for odd s and NP-hard for even s [11]. The Maximum 2-Club problem has been well-studied from both parameterized complexity and exact algorithms perspectives. It is possible to design a branching algorithm with running-time $O(2^{n-\ell} \cdot poly(n))$ for the dual parameter $n - \ell$ [24]. It is proved in [13] that, unless the Strong Exponential Time Hypothesis fails, such an algorithm cannot be improved to $O((2 - \epsilon)^{n-\ell} \cdot nm)$ for any $\epsilon > 0$. Chang et al. show that the s-club problem can be solved in $O(1.62^n)$-time [4]. Hartung, Komusiewicz and Nichterlein [13] proved that a maximum 2-club can be found in time $2^{O(2^{tw})} \cdot n^2$ where tw denotes the treewidth of the input graph. They also presented an algorithm with the same complexity (up to a constant in the exponent) regarding the larger parameter *vertex cover size*. Then in [14], Hartung, Komusiewicz, Nichterlein and Suchý

provide double-exponential FPT algorithms for two other parameters: the vertex deletion distance to cluster graphs (i.e. graphs in which every connected component induces a clique), and for vertex deletion to cographs (graphs that do not admit a path on four vertices as an induced subgraph).

Our results

We investigate the Maximum 2-Club problem from both algorithmic and structural aspects. We first consider structural parameterizations of the problem. We provide evidence that the algorithm by Hartung, Komusiewicz and Nichterlein [13], whose dependency on the vertex cover size $\mathrm{vc}(G)$ is $2^{O(2^{\mathrm{vc}})}$ is essentially the best one can hope for assuming the Exponential Time Hypothesis holds. Our result actually holds even for split graphs (the vertex set of a split graph being partitionable into a maximal clique and an independent set, the size of a vertex cover is at most that of a maximum clique which implies the claimed result) and also implies a sub-exponential time lower bound on such graphs.

▶ **Theorem 1.** *If there exists an algorithm which, given as input a split graph $G = (V, E)$, computes the maximum size of a 2-club $S \subseteq V$ of G and runs in time $2^{2^{o(\omega(G))}}2^{o(|V|)}$, where $\omega(G)$ is the maximum size of a clique of G, then the ETH is false.*

As a consequence, the double-exponential algorithms parameterized by treewidth [13], by vertex deletion distance to cluster graphs, and by vertex deletion to cographs [14] are essentially tight, since the vertex cover size is greater than or equal to all other parameters.

We then turn our attention to exact algorithms and provide an improved exact algorithm for this problem on chordal graphs. Our algorithm relies on the simple observation that any 2-club that is not a clique must intersect a minimal separator. Then, by a careful analysis of the remaining possibilities, we observe that the problem can be reduced to that of finding a maximum-sized independent set in an auxiliary graph with the same number of vertices. Since all minimal separators of a chordal graph can be enumerated in linear time (see e.g. [17]) and since a maximum independent set in n-vertex graphs can be found in time $O^*(1.1996^n)$ [26], we obtain the following.

▶ **Theorem 2.** *The maximum 2-club of a chordal graph G can be found in time $O^*(1.1996^n)$.*

This reduction to Maximum Independent Set can also be used to enumerate all maximum (or inclusion-wise maximal) 2-clubs of a chordal graph in time $O^*(3^{\frac{n}{3}})$, using the enumeration algorithm of maximal independent sets of [18], with the same running time.

▶ **Theorem 3.** *Chordal graphs have $O^*(3^{\frac{n}{3}})$ maximal 2-clubs, enumerable within the same running time.*

Note that n-vertex graphs can have up to $3^{\frac{n}{3}}$ maximum (or inclusion-wise maximal) independent sets as shown by a disjoint union of $n/3$ triangles, and this upper bound is tight [18]. We prove that a similar situation occurs when counting/enumerating the maximum (or inclusion-wise maximal) 2-clubs of chordal graphs, or even split graphs, by showing that there exist split graphs having $\Omega^*(3^{\frac{n}{3}})$ maximum (or inclusion-wise maximal) 2-clubs. As a byproduct, the bound of Theorem 3 is tight.

▶ **Theorem 4.** *For any $n \in \mathbb{N}$, there exists a split graph G of n vertices that contains $\Omega(3^{\frac{n}{3}}/\operatorname{poly}(n))$ distinct maximum (and inclusion-wise maximal) 2-clubs, for some fixed polynomial* poly*.*

2 Preliminaries

We consider simple, undirected graphs $G = (V, E)$ where V denotes the vertex set of G and E its edge set. We sometimes use $V(G)$ and $E(G)$ to clarify the context. Given a vertex v of G, we let $N(v)$ denote the *open neighborhood* of v, that is $N(v) = \{u \in V : uv \in E\}$. The *closed neighborhood* of v, denoted $N[v]$ is defined as $N(v) \cup \{v\}$. Given a subset of vertices S of G, we let $N(S) = \cup_{s \in S} N(s) \setminus S$ denote the open neighborhood of S. The closed neighborhood of S is similarly defined as $N[S] = N(S) \cup S$. We sometimes add a reference to G in the notation, namely $N_G(v)$ and $N_G[v]$ to clarify the context (and similarly for subsets S). The subgraph *induced by* S is defined as $G[S] = (S, E_S)$ where $E_S = \{uv \in E : (u, v) \in S^2\}$. For simplicity let $G - S$ denote the graph $G[V \setminus S]$.

A path $P = (x_1, \ldots, x_p)$ is a sequence of distinct vertices such that every pair of successive vertices are adjacent. If moreover $x_1 x_p \in E$ then $(x_1, \ldots, x_p)$ is a cycle. The length of a path is its number of vertices, minus one. Given two vertices u and v of G the length of a shortest path between u and v in G is denoted as $dist_G(u, v)$. For the sake of simplicity, for a given subset of vertices S of $V(G)$ we use $dist_S(u, v)$ to denote $dist_{G[S]}(u, v)$. The diameter of a graph G is the maximum length of any of its shortest paths.

Throughout the paper we will consider several special subsets of vertices and graph classes. A *clique* is a set of pairwise adjacent vertices while an *independent set* is a set of pairwise non-adjacent vertices. Given a graph $G = (V, E)$ with $|V| = n$ vertices and an integer $1 \leqslant s \leqslant n$, a subset $S \subseteq V$ is an s-club if the diameter of $G[S]$ is at most s. In all cases the classical associated problems ask for a set of maximum cardinality with the desired property. A graph is a *split graph* if it can be partitioned into a clique and an independent set. A graph is *chordal* if every cycle with at least 4 vertices has a chord, that is, an edge between two non-consecutive vertices of the cycle.

A *vertex cover* of G is a set of vertices that intersects all the edges of the graph. The *vertex cover size* of graph G, denoted $\mathrm{vc}(G)$, is the minimum size of a vertex cover (sometimes, when it is clear from the context, we simply speak of the "vertex cover of G" to refer to the vertex cover size). The maximum size of a clique is denoted $\omega(G)$. In a split graph G the two parameters are strongly related, since $\omega(G) - 1 \leq \mathrm{vc}(G) \leq \omega(G)$.

In a graph $G = (V, E)$, a subset $S \subseteq V$ is a *separator* for two vertices u and v if removing S from G separates u and v in two distinct connected components. The set S is a *minimal separator* for u and v if moreover no proper subset of S separates u and v. We also say that S is a (u, v)-*separator*, respectively a (u, v)-*minimal separator.*

▶ **Lemma 5** (folklore). *Let S be a (u, v)-minimal separator and let C_u (res. C_v) be the connected component of $G - S$ containing u (resp. v). Then $N_G(C_u) = N_G(C_v) = S$.*

The following observation is a well-known fact about chordal graphs. Here, we simply call *minimal separator* of G a vertex set S that is a (u, v)-minimal separator for some pair (u, v) of vertices.

▶ **Observation 6** ([6]). *A graph G is chordal if and only if each minimal separator of G is a clique.*

The Exponential Time Hypothesis (ETH) formulated by Impagliazzo and Paturi [15] conjectures that the 3-SAT problem cannot be solved in sub-exponential time. The Strong Exponential Time Hypothesis (SETH) assumes that, for all $\delta < 1$, there exists a k such that k-SAT cannot be solved in $O(2^{\delta n})$ time [2]. In other words, this hypothesis implies that SAT cannot be solved in $O(2^{\delta n})$ time for any $\delta < 1$.

By standard reduction from 3-SAT to Maximum Independent Set (see also Corollary 11.10 in the book of Fomin and Kratsch [8]), we have that the latter problem cannot be solved in time $2^{o(n+m)}$ unless ETH fails.

▶ **Theorem 7.** *Unless ETH fails, there is no $2^{o(n+m)}$ algorithm for Maximum Independent Set.*

Finally, we use the O^* notation to indicate that polynomial factors are omitted from the expression. That is, we write $O^*(f(n))$ whenever the complexity is $O(f(n) \cdot poly(n))$. We also use the $\Omega^*(f(n))$ notation to refer to functions $\Omega(f(n)/\operatorname{poly}(n)))$ for some polynomial poly.

3 Lower Bound for Maximum 2-club parameterized by vertex cover

Recall that $\omega(G)$ denotes the maximum clique size of G which, if G is a split graph, can be computed in polynomial time. We show that computing the largest 2-club (i.e., solving problem Maximum 2-club) has a double-exponential dependence parameterized by the clique size of a split graph, assuming ETH. Because in a split graph the maximum clique size is almost equal to the minimum vertex cover ($\omega(G) - 1 \leq \text{vc}(G) \leq \omega(G)$), this implies that the problem has at least double-exponential parameter dependence when parameterized by vertex cover. Furthermore, because the theorem is based on a reduction from Maximum Independent Set with only a linear blow-up in size, we can also conclude by Theorem 7 that, even for split graphs, the problem cannot be solved in time sub-exponential in the order of the input graph.

▶ **Theorem 1.** *If there exists an algorithm which, given as input a split graph $G = (V, E)$, computes the maximum size of a 2-club $S \subseteq V$ of G and runs in time $2^{2^{o(\omega(G))}} 2^{o(|V|)}$, where $\omega(G)$ is the maximum size of a clique of G, then the ETH is false.*

Proof. We present a reduction from Maximum Independent Set. Suppose we are given a graph $G = (V, E)$ on n vertices and m edges and are asked if G has an independent set of size at least s. We will construct a split graph G' and a target size s' with the following properties:

1. G' has a 2-club of size at least s' (that is, a set of vertices S with $|S| \geq s'$ and the diameter of $G'[S]$ being at most 2) if and only if G has an independent set of size s.
2. G' is split and its maximum clique size is $\omega(G') = O(\log n)$.
3. $|V(G')| = O(n + m)$ and the construction can be carried out in polynomial time.

Let us first observe that if we achieve the above, we obtain the theorem. Indeed, we can decide if G has an independent set of size s by constructing G' and executing the supposed algorithm for finding a maximum 2-club. The running time would be dominated by the execution of this algorithm, which would take $2^{2^{o(\omega(G'))}} 2^{o(n+m)} = 2^{o(n+m)}$. We would therefore have a sub-exponential algorithm for Maximum Independent Set, which would contradict the ETH (Theorem 7).

Let us now describe the construction of G'. We assume without loss of generality that n and m are powers of 2 (otherwise we can take the union of G with an appropriate number of vertex-disjoint copies of K_2 and K_1 adjusting s as needed, while increasing the size of G by at most a constant factor). The vertices of G' are partitioned into a clique C and an independent set I. More precisely:

- The vertices of the clique C are partitioned into three sets X, Y, Z, with $|X| = |Y| = 2\log n + 1$ and $|Z| = 2\log m$. Furthermore, we add to the clique two more vertices, call them z_0, z_1.
- For each vertex $v \in V$ we construct a vertex a_v and place it in the independent set I of G'. Let $A = \{a_v \mid v \in V\}$.
- For each edge $e \in E$ with $e = uv$, with $u, v \in V$, we construct two vertices b_{eu} and b_{ev} and place them in the independent set I of G'. Let $B = \{b_{eu} \mid e \in E, u \in e\}$.

At this point the graph has a clique of size $4\log n + 2\log m + 4$ and an independent set of size $n + 2m$ and what remains is to describe the edges between C and I. Before we proceed, recall the basic fact that for all positive integers k we have $\binom{2k}{k} \geq 2^k$.

We define three auxiliary functions f_X, f_Y, f_Z as follows. First $f_X : V \to 2^X$ is a function which assigns to each $v \in V$ a distinct set $f_X(v) \subseteq X$ such that $|f_X(v)| = \log n + 1$. Observe that since $|X| = 2\log n + 1$, there exist at least $\binom{2\log n+1}{\log n+1} = \binom{2\log n+1}{\log n} \geq \binom{2\log n}{\log n} \geq 2^{\log n} = n$ distinct such sets, so it is possible to make f_X injective; furthermore we can construct such a function in time polynomial in n. Similarly, $f_Y : V \to 2^Y$ is a function that assigns to each $v \in V$ a distinct set $f_Y(v) \subseteq Y$ with $|f_Y(v)| = \log n + 1$. In a similar fashion, $f_Z : E \to 2^Z$ is a function which assigns to each $e \in E$ a distinct set $f_Z(e) \subseteq Z$ with $|f_Z(e)| = \log m$.

We are now ready to describe the remaining edges of G':

1. For each $v \in V$ make a_v adjacent to all of $f_X(v)$ and all of $f_Y(v)$.
2. For each $e \in E$ with $e = uv$, arbitrarily select one of u, v, say u, to be the first endpoint of e. Make b_{eu} adjacent to $\{z_0\} \cup f_Z(e) \cup (X \setminus f_X(u))$ and make b_{ev} adjacent to $\{z_1\} \cup (Z \setminus f_Z(e)) \cup (Y \setminus f_Y(v))$.

This completes the construction of G' and we set $s' = s + m + |C|$. Before proving that G' has a 2-club of size s' if and only if G has an independent set of size s, it will be helpful to make some basic observations about the independent set I of G'.

▷ Claim 8. The only pairs of vertices of I which are at distance 3 in G' are of the following two forms:

1. b_{eu}, b_{ev}, for some $e \in E$ with $e = uv$
2. b_{ev}, a_v, for some $e \in E$ with $v \in E$.

Proof. It is easy to see that pairs of the stated forms indeed have distance 3: b_{eu}, b_{ev} have disjoint neighborhoods in Z by construction; b_{ev}, a_v have disjoint neighborhoods because a_v has no neighbor in Z, and its neighborhood in X is $f_X(v)$ while b_{ev} is at most adjacent to $X \setminus f_X(v)$ (and similarly for Y).

What is more interesting is to observe that these are the only pairs of vertices from I at distance 3. To consider all other possible pairs we have the following cases:

1. Two vertices $a_v, a_u \in A$ are at distance 2. This can be seen as $|f_X(v)| = |f_X(u)| > \frac{|X|}{2}$ so $f_X(u)$ and $f_X(v)$ cannot be disjoint.
2. Two vertices $b_{ev}, b_{fu} \in B$, for two distinct edges $e, f \in E$ are at distance 2. This is easy to see if v, u are the first endpoints of the edges e, f, as they are both adjacent to z_0 (and similarly if they are both the second endpoints, they are both adjacent to z_1). Suppose then that v is the first endpoint of e and u the second endpoint of f. However, $f_Z(e) \neq f_Z(f)$, since f_Z is injective. Because $|f_Z(e)| = |f_Z(f)|$, there exist w with $w \in f_Z(e)$ and $w \notin f_Z(f)$. We conclude that w is a common neighbor of b_{ev}, b_{fu}.
3. A vertex $a_v \in A$ and a vertex $b_{eu} \in B$, with $u \neq v$ are at distance 2. Suppose without loss of generality (the other case being symmetric) that u is the first endpoint of e. Then b_{eu} is adjacent to $X \setminus f_X(u)$, therefore since all vertices of A have $\log n + 1$ neighbors in X, the only vertex of A that has no common neighbor with b_{eu} is a_u.

This concludes the proof of Claim 8. ◁

Armed with Claim 8, we can now establish the correctness of the reduction. First, if G has an independent set S of size s, we select a 2-club S' in G' as follows: for each $v \in S$ we place a_v in S'; we place all of the clique in S'; and for each $uv \in E$ at least one of its endpoints, say u, is not in S, so we place the corresponding vertex b_{eu} in S'. Clearly we have a set of the desired size, that is $s' = s + m + |C|$. To see that it is a 2-club we observe that since all of the clique is in S' the distances between any two vertices are preserved, and we have not selected any of the forbidden pairs of Claim 8.

For the converse direction, suppose we have a 2-club S' of size s' in G'. We can assume without loss of generality that S' contains C because vertices of C are at distance at most 2 from all other vertices of the graph. Furthermore, we can modify S' so that it contains exactly m vertices of B as follows: if for some $e \in E$ with $e = uv$ neither of b_{eu}, b_{ev} is contained in S', then we add b_{eu} to S' and remove a_u (if necessary), which thanks to Claim 8 gives another solution of at least the same size. Since for any edge $e \in E$ with $e = uv$ we cannot have both b_{eu} and b_{ev} in S', as by Claim 8 these vertices are at distance 3, we conclude that S' contains s vertices of A, from which we extract a set S of G of size s. We claim that S is an independent set. Indeed, suppose $u, v \in S$ and $e = uv \in E$, then S' must contain b_{eu} or b_{ev}. Say without loss of generality that $b_{eu} \in S'$. Since $a_u \in S'$ (as we placed $u \in S$), by Claim 8 S' contains two vertices at distance 3, contradiction. We conclude that if G' has a 2-club of size s', then G has an independent set of size s. ◀

4 An exact algorithm for Maximum 2-club in chordal graphs

We now turn our attention to chordal graphs with the aim of proposing an improved exact algorithm for Maximum 2-club for this class. Our algorithm relies on the simple observation that any 2-club that is not a clique must intersect a minimal separator (Observation 10). Then, by a careful analysis of the remaining possibilities, we observe that the problem can be reduced to that of finding a maximum-sized independent set in an arbitrary graph with the same number of vertices.

Throughout this section we let $G = (V, E)$ be a chordal graph, and M be a maximum 2-club of G. We point out that the same results hold for inclusion-wise maximal 2-clubs. The following observation comes from the fact that a maximum clique can be found in linear time in chordal graphs through the enumeration of all minimal separators (see e.g. [17]).

▶ **Observation 9.** *If M is a clique, it can be found in time $O(n + m)$.*

In the following, we will study the case in which M is not a clique.

▶ **Observation 10.** *There exist $x, y \in M$ and a (x, y)-minimal separator S so that $S \cap M \neq \emptyset$.*

Proof. As M is not a clique, there exist $x, y \in M$ so that $dist_M(x, y) \geq 2$. As M is a 2-club, we actually have $dist_M(x, y) = 2$, and there exists $z \in M$ so that (x, z, y) is an induced path in M. Then, for any (x, y)-separator S, $z \in S$ and thus $S \cap M \neq \emptyset$. ◀

We now fix such a set S and provide further observations with respect to S.

▶ **Lemma 11.** *The maximum 2-club M is included in the closed neighborhood of S in G.*

Proof. By contradiction, let us assume that $z \in M \setminus N_G[S]$ exists. As S is a (x, y)-separator, x and y are in two different connected components of $G - S$. The vertex z is also in a connected component of $G - S$, so at least one among x, y are in a different component than z. Assume w.l.o.g. that z is not in the connected component of $G - S$ containing x. We

claim that $dist_G(x,z) \geq 3$. Indeed if $dist_G(x,z) \leq 2$ there must be some path (x,w,z) in G, from the component C_x of $G-S$ containing x to the component C_z containing z. This path necessarily intersects $N_G(C_x)$, and by Lemma 5 we have $N_G(C_x) = S$. Therefore $w \in S$, contradicting the assumption that z is not in the closed neighborhood of S. It remains that $dist_G(x,z) \geq 3$, and M is not a 2-club. ◀

▶ **Definition 12** (bad pair). *Let C be a connected component of $G-S$ and consider two vertices $a,b \in C \cap N(S)$. We say that (a,b) is a bad pair if $dist_G(a,b) \geq 2$ and $N(a) \cap N(b) \cap S = \emptyset$.*

▶ **Lemma 13.** *Let $(a,b) \in M^2$ a bad pair. For all $c \in (N(a) \cap N(b)) \setminus S$, we have that $N(a) \cap S \subseteq N(c)$ and $N(b) \cap S \subseteq N(c)$.*

Proof. By contradiction, let us assume that there exists $a' \in N(a) \cap S$ so that $a' \notin N(c)$. As $b \in N(S)$ by definition of bad pairs, there exists $b' \in N(b) \cap S$, different from a'. If $b' \in N(c)$ then (a,a',b',c) is a chordless C_4 and otherwise, (a,a',b',b,c) is a chordless C_5. Then, G cannot be chordal. It remains that $N(a) \cap S \subseteq N(c)$, and by symmetry also $N(b) \cap S \subseteq N(c)$. ◀

▶ **Lemma 14.** *Let a,b be two vertices in $N[S]$. If the distance between a and b in $G[N[S]]$ is at least 3, then the distance between a and b in G is also at least 3.*

Proof. Since S is a clique (by Observation 6), note that the distance $dist_{N[S]}(a,b)$ between a and b in $G[N[S]]$ is also at most 3.

Firstly, assume that a and b are in a same connected component C of $G-S$. Then the fact that the distance $dist_{N[S]}(a,b) = 3$ implies that (a,b) is a bad pair. Suppose that $dist_G(a,b) \leq 2$. The only possibility would be that a and b have a common neighbor c in $V(G) \setminus N[S]$. Then by Lemma 13, $N(a) \cap S \subseteq N(c)$, thus $c \in N[S]$ – a contradiction.

It remains the case when a,b are in different components of $G-S$. But then if $dist_G(a,b) \leq 2$ the common neighbor of a and b on the (a,b)-path of length 2 is necessarily in S, contradicting $dist_{N[S]}(a,b) = 3$. ◀

▶ **Definition 15** (auxiliary graph for S). *Let us define the* auxiliary graph H for the minimal separator S *as follows: $V(H) = N_G[S]$ and for any pair of vertices $a,b \in N_G[S]$, $ab \in E(H)$ if and only if $dist_G(a,b) = 3$.*

Again, recall that since S is a clique, the distance between any pair of vertices of $N_G[S]$ in G is at most 3. Also, by Lemma 14, being at distance 3 in G is the same as being at distance 3 in $G[N_G[S]]$. The following observation is simple but crucial in proving that the maximum (or maximal) 2-club M is actually a maximum (maximal) independent set of H.

▶ **Lemma 16.** *Let I be an independent set of H, (a,b) a bad pair such that $a,b \in I$, and $c \in ((N_G(a) \cap N_G(b) \cap N_G(S)) \setminus I$. Then, $I \cup \{c\}$ is an independent set of H.*

Proof. By contradiction, let us assume that there exists $d \in I$ so that $dist_G(c,d) \geq 3$. Since $a,b,d \in I$, we also have $dist_G(a,d) \leq 2$, $dist_G(b,d) \leq 2$ by construction of the auxiliary graph H (Definition 15). Then d is in the same connected component of $G-S$ as a,b and c (otherwise we cannot have $dist_G(a,d) \leq 2$, $dist_G(b,d) \leq 2$ and $dist_G(c,d) \geq 3$ simultaneously according to Lemma 13). Also, (a,d) is a bad pair: we cannot have $dist_G(a,d) = 1$ because it would imply that $dist_G(c,d) \leq 2$ since $ac \in E(G)$, and we cannot have $dist_G(a,d) = 2$ and a,d have a common neighbor c' in S, because Lemma 13 applied to the bad pair (a,b) implies that $cc' \in E(G)$ thus $dist_G(c,d) \leq 2$. Similarly, (b,d) is a bad pair. Then there exist $e,f \in G$ so that (d,e,a) and (d,f,b) are induced paths, and e and f are in the same connected component as a,b,c and d (possibly $e = f$).

Assume that $e \neq f$. As G is chordal, there exists at least one of the chords (c, e) and (c, f) in the cycle (a, c, b, f, d, e) . Then, $dist_G(c, d) = 2$, which contradicts our premise. If $e = f$ then the cycle (a, c, b, f) must have a chord, and the only possibility is cf, entailing again the contradiction $dist_G(c, d) = 2$. This concludes the proof. ◀

▶ **Lemma 17.** *Let I be a maximal independent set of H. Then $S \subseteq I$ and I is a maximal 2-club of G among those contained in $N_G[S]$.*

Proof. Since S is a clique in G (Observation 6), by maximality of I and definition of H we have $S \subseteq I$. Let us start by showing that I is a 2-club in G. To that aim, we prove that for any pair of vertices $(a, b) \in I^2$ it holds that $dist_I(a, b) \leqslant 2$. (In the remaining of the proof all distances are considered in G.) Let $(a, b) \in I^2$. If $a, b \in S$, $dist_I(a, b) = 1$ since S is a clique and we are done. If $a \in I \setminus S$ and $b \in S$, $dist_I(a, b) \leq 2$ by definition of H (recall that $V(H) = N_G[S]$ and that $a \in N_G[S]$).
Let us assume $a, b \in I \setminus S$. If (a, b) is not a bad pair (Definition 12), then either $dist_G(a, b) = 1$ or there exists $c \in S$ so that $c \in N_G(a) \cap N_G(b)$ and thus $dist_I(a, b) = 2$. If (a, b) is a bad pair, $N_G(a) \cap N_G(b) \cap S = \emptyset$. As $dist_G(a, b) \leq 2$ (by definition of H and the fact that $a, b \in I$), by Lemma 14 we also have $dist_{N_G[S]}(a, b) \leq 2$. Thus there exists $c \in N_G(a) \cap N_G(b) \cap N_G(S)$. If $c \notin I$, Lemma 16 shows that $I \cup \{c\}$ is an independent set of H. As I is maximal, this is not possible. Thus, $c \in I$, (a, c, b) is an induced path in I and $dist_I(a, b) = 2$.

To prove the maximality, assume there is some 2-club M of G contained in $N_G[S]$, strictly containing I. By construction of H, since any pair of vertices of M are at distance at most 2 in $G[M]$ (and hence in G), we have that M is an independent set of H, contradicting the maximality of I. ◀

The converse of Lemma 17 also holds.

▶ **Lemma 18.** *Let M be a 2-club contained in $N_G[S]$. Then M is an independent set of the auxiliary graph H.*

Proof. Since M is a 2-club any two vertices of M are at distance at most 2 in G, thus they are non-adjacent in H. ◀

▶ **Theorem 2.** *The maximum 2-club of a chordal graph G can be found in time $O^*(1.1996^n)$.*

Proof. We use the following strategy to find a maximum 2-club in a chordal graph:

1. we enumerate all minimal separators of G, which can be done in time $O(n + m)$ (see e.g. [17]);
2. for every such separator S,
   a. we construct the auxiliary graph H for S defined as $(N_G[S], E_H)$ where $ab \in E(H)$ iff $dist_G(a, b) \geq 3$;
   b. we apply the algorithm of Xiao and Nagamochi [26] to compute a maximum independent set I_S of H in time $O(1.1996^n)$;
3. let I be the largest set I_S computed among all sets S.
4. let K be the largest clique of G, which can be obtained in linear time, e.g. by [20].
5. we output the largest set among I and K.

Observe that the set M returned by the algorithm is indeed a 2-club; this is trivial if the algorithm returns a clique K, and comes from Lemma 17 if the set returned by the algorithm is $I = I_S$ for some minimal separator S.

Let M' be a maximum 2-club of G, we need to prove that $|M| \geq |M'|$. If M' is a clique, then the maximum clique K constructed by the algorithm is at least of the same size and the conclusion follows. If M' is not a clique then it is contained in the neighborhood $N_G[S]$ of some minimal separator S of G by Lemma 11. Also by Lemma 18, it is an independent set of the auxiliary graph H for the minimal separator S. Therefore the set I_S constructed by the algorithm is of size at least M', and so is M. ◀

We emphasise that the reduction of Lemma 17 can also be used to enumerate all maximal 2-clubs in a chordal graph in time $O^*(3^{\frac{n}{3}})$.

▶ **Theorem 3.** *Chordal graphs have $O^*(3^{\frac{n}{3}})$ maximal 2-clubs, enumerable within the same running time.*

Proof. Let G be a chordal graph. Maximal 2-clubs that are cliques are also maximal cliques of G. Their number is at most n and they are enumerable in linear time in chordal graphs [17].

It remains to enumerate maximal 2-clubs M that are not cliques. Recall that such 2-clubs intersect some minimal separator S of G (Lemma 11). Fix a minimal separator S, we count and enumerate the maximal 2-clubs M intersecting S. Construct the graph H associated to S as in Definition 15, clearly M is an independent set of H (Lemma 18). By Lemma 17 and the maximality of M, we actually must have that M is a maximal independent set of H. By a classic result of Moon and Moser [18], the graph H with at most n vertices has at most $3^{\frac{n}{3}}$ maximal independent sets, enumerable in time $O^*(3^{\frac{n}{3}})$.

Recall that G has at most n minimal separators which can be enumerated in linear time [17], completing the proof of the statement. ◀

5 Split graphs with $\Omega^*(3^{\frac{n}{3}})$ maximum 2-clubs

In this section we provide a construction of (split) graphs that have $\Omega^*(3^{\frac{n}{3}})$ maximum 2-clubs, where $3^{\frac{n}{3}} \approx 1.4423^n$.

To that aim, we recursively construct the set of split graphs $(G_k = ((C_k, I_k), E_k))_{k \in \mathbb{N}^*}$, such that for all $k \in \mathbb{N}^*$, $|C_k| = 6k - 3$ and $|I_k| = 3^k$, so the graph G_k has $n = 3^k + O(k)$ vertices. Moreover the independent set I_k is partitioned into 3^{k-1} triplets. Then we will show that G_k has at least $3^{3^{k-1}}$ maximum (and inclusion-wise maximal) 2-clubs obtained by picking exactly one vertex from each triplet, plus the set C_k. This will lead to the $\Omega^*(3^{\frac{n}{3}})$ lower bound on the number of maximum (or inclusion-wise maximal) 2-clubs.

Construction of graphs G_k

We refer the reader to Figure 1 for an explanation of the construction.

- C_1 is a clique of size 3 and $I_1 = \{x_1, y_1, z_1\}$ is an independent set of size 3. We give each vertex of I_1 one neighbor in C_1 such that $(N(x_1), N(y_1), N(z_1))$ is a partition of C.
- $C_{k+1} = C_k \cup \{a_{k+1}, b_{k+1}, c_{k+1}, d_{k+1}, e_{k+1}, f_{k+1}\}$ with $\{a_{k+1}, b_{k+1}, c_{k+1}, d_{k+1}, e_{k+1}, f_{k+1}\}$ six new vertices. The independent set I_k is comprised of 3^k vertices split in 3^{k-1} distinct triplets (x_i, y_i, z_i) with $i \in [\![1, 3^{k-1}]\!]$. To construct I_{k+1}, we triple the size of I_k by replacing each original vertex $v \in I_k$ with three copies v^1, v^2, v^3. Thus, for each $i \in [\![1, 3^{k-1}]\!]$, we transform the triplet (x_i, y_i, z_i) into three triplets (x_i^1, y_i^1, z_i^1), (x_i^2, y_i^2, z_i^2) and (x_i^3, y_i^3, z_i^3), keeping the invariant that I_{k+1} is formed of 3^k triplets (which can arbitrarily be renumbered from 1 to 3^k). Each edge uv of G_k with $u \in C_k$ and $v \in I_k$ will also be tripled, i.e., we put in E_{k+1} the three edges uv^1, uv^2 and uv^3.
  Eventually, for all $i \in [\![1, 3^{k-1}]\!]$ we add the following 18 edges to E_{k+1}:

- $(a_{k+1}, x_i^1), (a_{k+1}, y_i^2), (a_{k+1}, z_i^3)$
- $(b_{k+1}, y_i^1), (b_{k+1}, z_i^2), (b_{k+1}, x_i^3)$
- $(c_{k+1}, z_i^1), (c_{k+1}, x_i^2), (c_{k+1}, y_i^3)$
- $(d_{k+1}, x_i^1), (d_{k+1}, z_i^2), (d_{k+1}, y_i^3)$
- $(e_{k+1}, y_i^1), (e_{k+1}, x_i^2), (e_{k+1}, z_i^3)$
- $(f_{k+1}, z_i^1), (f_{k+1}, y_i^2), (f_{k+1}, x_i^3)$

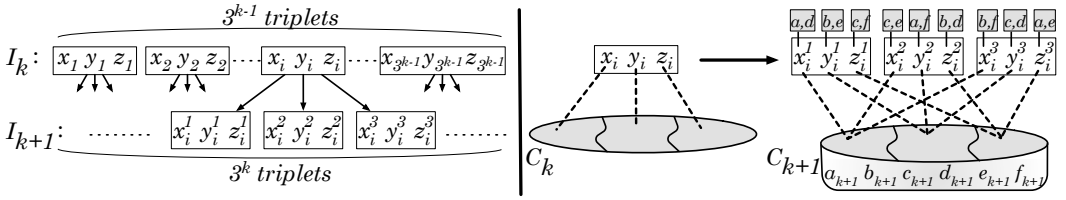


Figure 1 Illustration of the construction. On the left side: each triplet from I_k is replaced by 3 triplets in I_{k+1}. On the right side: the neighborhoods (represented by dashed edges) of x_i, y_i and z_i form a partition of C_k. The clique C_{k+1} is made from C_k together with new vertices $a_{k+1}, b_{k+1}, c_{k+1}, d_{k+1}, e_{k+1}, f_{k+1}$. These 6 vertices of C_{k+1} are connected to the triplets of I_{k+1} as represented above the triplets (where subscripts are forgotten). In particular, the vertices a_{k+1} and d_{k+1} insure that x_i^1 is at distance at most 2 from all the other vertices in I_{k+1} except for y_i^1 and z_i^1.

▶ **Lemma 19.** *Let $k \geq 2$ be an integer, let $i \neq j$ be two integers in $[\![1, 3^{k-1}]\!]$, and let (x_i, y_i, z_i) and (x_j, y_j, z_j) denote the ith and the jth triplets of I_k. For all $u \in \{x_i, y_i, z_i\}$, $v \in \{x_j, y_j, z_j\}$, $N_{G_k}[u] \cap N_{G_k}[v] \neq \emptyset$.*

Proof. By construction of G_k and the set of 18 edges added between the six vertices $\{a_k, b_k, c_k, d_k, e_k, f_k\}$ of C_k and the triplets of I_k, we ensured that u, v have a common neighbor among $\{a_k, b_k, c_k, d_k, e_k, f_k\}$. See also Figure 1. ◀

▶ **Lemma 20.** *Let $k \in \mathbb{N}^*$. For all $i \in [\![1, 3^{n-1}]\!]$, let (x_i, y_i, z_i) denote the ith triplet of I_k. For all $u, v \in \{x_i, y_i, z_i\}$, $N_{G_k}[u] \cap N_{G_k}[v] = \emptyset$.*

Proof. Let us prove this recursively. By definition of G_1, $N_{G_1}[u] \cap N_{G_1}[v] = \emptyset$. Then, let us assume the premise is true for k. When tripling I_k to create I_{k+1}, we get $u^1, v^1, u^2, v^2, u^3, v^3$ from u and v. For $\ell \in [\![1, 3]\!]$, $N_{I_{k+1}}[u^\ell] = N_{I_{k+1}}[v^\ell] = \emptyset$ based on the construction of $(G_k)_{k \in \mathbb{N}^*}$. Indeed u and v have no common neighbor in C_k in graph G_k therefore u^ℓ and v^ℓ will have no common neighbor in $C_k \subseteq C_{k+1}$ in graph G_{k+1}. Amongst the six new vertices added to C_{k+1}, none is simultaneously a neighbor of u^ℓ and v^ℓ. Thus, $N_{G_{k+1}}[u^\ell] \cap N_{G_{k+1}}[v^\ell] = \emptyset$. ◀

▶ **Lemma 21.** *Let $k \in \mathbb{N}^*$. Let $A \subseteq I_k$ be a set of size $\frac{|I_k|}{3}$ such that one and only one vertex of each triplet of I_k is in A. Then, $C_k \cup A$ is a maximum 2-club of G_k.*

Proof. Let us show that $C_k \cup A$ is a 2-club of G_k.
Let $u, v \in C_k \cup A$. If $u, v \in C_k$, $dist_{C_k \cup A}(u, v) = 1$ as C_k is a clique. If $u \in C_k$, $v \in A$, $dist_{C_k \cup A}(u, v) \leq 2$ as C_k is a clique and $N_{G_k}[v] \cap C_k \neq \emptyset$ by definition of G_k. If $u, v \in A$, we know by definition of A that u and v are not in the same triplet of I_k. Then u and v have a common neighbor in C_k (Lemma 19) and thus $dist_{C_k \cup A}(u, v) = 2$.

Now, let us prove that $C_k \cup A$ is maximum. Lemma 20 shows that for u, v in the same triplet of I_k, $dist_{G_k}(u, v) \geq 3$. Thus, there cannot exist a 2-club in G_k that has more vertices of I_k than 3^{k-1} (i.e. more than one by triplet). As for all $u \in G_k \setminus I_k = C_k$, $u \in C_k \cup A$, so $C_k \cup A$ is maximum. ◀

▶ **Theorem 4.** *For any $n \in \mathbb{N}$, there exists a split graph G of n vertices that contains $\Omega(3^{\frac{n}{3}}/\operatorname{poly}(n))$ distinct maximum (and inclusion-wise maximal) 2-clubs, for some fixed polynomial* poly*.*

Proof. Let k be the maximum integer such that $3^k+6k+3 \leq n$, in particular $3^k \geq n-O(\log n)$ hence $3^{k-1} \geq n/3 - O(\log n)$.

We focus on graph G_k, which has at most n vertices. (We can complete it with isolated vertices to obtain a graph with exactly n vertices, without diminishing the number of maximum 2-clubs.) Recall that I_k is composed of 3^{k-1} triplets (as $|I_k| = 3^k$), and by Lemma 21 there are $3^{3^{k-1}}$ maximum 2-clubs in G_k, by picking one vertex in each triplet plus the set C_k.

Plugging in the observation that $3^{k-1} \geq n/3 - O(\log n)$, we have that $3^{3^{k-1}}$ is $\Omega^*(3^{\frac{n}{3}})$. Thus we have constructed a family of graphs with $\Omega^*(3^{\frac{n}{3}})$ maximum (and hence, inclusion-wise maximal) 2-clubs. ◀

▶ **Observation 22.** *The exact same reasoning can be used with subsets of size x different than 3, to obtain split graphs with $\Omega^*(x^{\frac{n}{x}})$ maximum 2-clubs. But the function $f : x \mapsto x^{\frac{1}{x}}$ is greatest for $x = 3$ amongst all $x \in \mathbb{N}$.*

6 Conclusion and open problems

We showed in this paper that the FPT algorithm for Maximum 2-club parameterized by vertex cover [13], with a double-exponential on $\operatorname{vc}(G)$, is essentially optimal under ETH. Note that [13] actually provides an algorithm with the same dependency on the treewidth of G, and since the latter is at most the vertex cover it cannot be essentially improved. A similar observation holds for the parameters vertex deletion distance to cographs and cluster graphs, respectively [14].

Consider problem Maximum 2-club, formulated as "does the input graph contain a 2-club of size at least ℓ", with parameter ℓ. The problem is FPT, with a simple argument which actually yields a Turing kernel [24]: if the input graph G has a vertex x of degree at least $\ell - 1$, then $N_G[x]$ is a trivial yes-instance. Therefore, we may assume that G has maximum degree $\ell - 1$, and seek, for each vertex x, a 2-club containing x – therefore we only work in the graph $G[N_G[N_G[x]]$, the second neighborhood of x. Because of the maximum degree being $\ell - 1$, such a graph has $O(\ell^2)$ vertices, so the whole algorithm runs in time $O^*(2^{O(\ell^2)})$. We re-ask the following natural question:

▶ **Question 1** ([14, 24])**.** *Is there an algorithm for finding a 2-club of size at least ℓ, parameterized by ℓ, with running time $O^*(2^{O(\ell)})$?*

When it comes to exact algorithms, we provided an algorithm for chordal graphs running in time $O^*(1.1996^n)$. For arbitrary graphs, the best known algorithm [24] runs in time $O^*(1.62^n)$ based on a simple and natural branching as follows. Let G^2 be the square of G, obtained from G by adding an edge between every pair of vertices at distance at most 2. If G^2 is a clique, then the whole graph is a 2-club. Otherwise, let x be a vertex and y be a non-neighbor of x in G^2. Branch on $G - x$ to find the largest 2-club not containing x, and also on $G - y$ to find the largest 2-club containing x and hence not containing y. To follow the terminology of [8], this corresponds to a branching vector $(2, 1)$: we reduce the number of vertices on which we can branch by 2 in the second case, since x is in the club and y is not, and by 1 in the first case when we simply discard x. This leads to the running time of $O^*(1.62^n)$. If, for example, x has two or more non-neighbors in G^2, then when x is in the 2-club we exclude at least 2 other vertices and the branching vector becomes $(3, 1)$, leading

to a better running time. In other words, the hardest instances are those where G^2 is a clique minus a matching. Not only such instances are easy to construct, but one can also rule out some naive ideas such as 'maybe in these instances, if y is the unique non-neighbor of x in G^2, then any maximum 2-club contains at least one among x, y'. Therefore we ask:

▶ **Question 2.** *Is there an exact algorithm for* Maximum 2-club *running in time* $O^*(c^n)$ *for some constant* $c \leq 1.61$*?*

Eventually, what is the maximum number of maximum (or inclusion-wise maximal) 2-clubs in an n-vertex graph?

▶ **Question 3.** *Is there a constant* $c > 3^{\frac{1}{3}} \simeq 1.4423$ *such that for infinitely many* $n \geq 0$ *there are* n*-vertex graphs with at least* c^n *maximum (or inclusion-wise maximal) 2-clubs?*

References

1 Yuichi Asahiro, Eiji Miyano, and Kazuaki Samizo. Approximating maximum diameter-bounded subgraphs. In *Latin American Symposium on Theoretical Informatics*, pages 615–626. Springer, 2010. doi:10.1007/978-3-642-12200-2_53.

2 Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. The complexity of satisfiability of small depth circuits. In *International Workshop on Parameterized and Exact Computation*, pages 75–85. Springer, 2009. doi:10.1007/978-3-642-11269-0_6.

3 Antoine Castillon. *Sous-graphes denses: algorithmes et analyse de complexité de certains problèmes de recherche et de couverture*. PhD thesis, Université de Lille, 2024.

4 Maw-Shang Chang, Ling-Ju Hung, Chih-Ren Lin, and Ping-Chen Su. Finding large-clubs in undirected graphs. *Computing*, 95(9):739–758, 2013. doi:10.1007/S00607-012-0263-3.

5 Brent N. Clark, Charles J. Colbourn, and David S. Johnson. Unit disk graphs. *Discret. Math.*, 86(1-3):165–177, 1990. doi:10.1016/0012-365X(90)90358-O.

6 Gabriel Andrew Dirac. On rigid circuit graphs. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 25:71–76, 1961.

7 Rodney G Downey and Michael Ralph Fellows. *Parameterized complexity*. Springer Science & Business Media, 2012.

8 Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2010. doi:10.1007/978-3-642-16533-7.

9 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

10 Michelle Girvan and Mark E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, 2002. doi:10.1073/pnas.122653799.

11 Petr A Golovach, Pinar Heggernes, Dieter Kratsch, and Arash Rafiey. Finding clubs in graph classes. *Discrete Applied Mathematics*, 174:57–65, 2014. doi:10.1016/J.DAM.2014.04.016.

12 Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer, 1988. doi:10.1007/978-3-642-97881-4.

13 Sepp Hartung, Christian Komusiewicz, and André Nichterlein. Parameterized algorithmics and computational experiments for finding 2-clubs. In *International Symposium on Parameterized and Exact Computation*, pages 231–241. Springer, 2012. doi:10.1007/978-3-642-33293-7_22.

14 Sepp Hartung, Christian Komusiewicz, André Nichterlein, and Ondřej Suchỳ. On structural parameterizations for the 2-club problem. *Discrete Applied Mathematics*, 185:79–92, 2015. doi:10.1016/J.DAM.2014.11.026.

15 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.

16 Christian Komusiewicz. Multivariate algorithmics for finding cohesive subnetworks. *Algorithms*, 9(1):21, 2016. doi:10.3390/A9010021.

17 P Sreenivasa Kumar and CE Veni Madhavan. Minimal vertex separators of chordal graphs. *Discrete Applied Mathematics*, 89(1-3):155–168, 1998. doi:10.1016/S0166-218X(98)00123-1.

18 John W. Moon and Leo Moser. On cliques in graphs. *Israel Journal of Mathematics*, 3(1):23–28, 1965. doi:10.1007/BF02760024.

19 Jaroslav Nesetril, Patrice Ossona de Mendez, Michal Pilipczuk, and Xuding Zhu. Clustering powers of sparse graphs. *Electron. J. Comb.*, 27(4):4, 2020. doi:10.37236/9417.

20 Donald J. Rose, R. Endre Tarjan, and George S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput.*, 5:266–283, 1976. doi:10.1137/0205021.

21 Bill Rosgen and Lorna Stewart. Complexity results on graphs with few cliques. *Discret. Math. Theor. Comput. Sci.*, 9(1), 2007. doi:10.46298/DMTCS.387.

22 Doron Rotem and Jorge Urrutia. Finding maximum cliques in circle graphs. *Networks*, 11(3):269–278, 1981. doi:10.1002/NET.3230110305.

23 Alexander Schäfer. *Exact algorithms for s-club finding and related problems*. PhD thesis, Friedrich-Schiller-University Jena, 2009.

24 Alexander Schäfer, Christian Komusiewicz, Hannes Moser, and Rolf Niedermeier. Parameterized computational complexity of finding small-diameter subgraphs. *Optimization Letters*, 6:883–891, 2012. doi:10.1007/S11590-011-0311-5.

25 David R. Wood. On the maximum number of cliques in a graph. *Graphs Comb.*, 23(3):337–352, 2007. doi:10.1007/S00373-007-0738-8.

26 Mingyu Xiao and Hiroshi Nagamochi. Exact algorithms for maximum independent set. *Information and Computation*, 255:126–146, 2017. doi:10.1016/J.IC.2017.06.001.

A Simple Algorithm for Combinatorial n-Fold ILPs Using the Steinitz Lemma

Sushmita Gupta
The Institute of Mathematical Sciences, HBNI, Chennai, India

Pallavi Jain
IIT Jodhpur, India

Sanjay Seetharaman
The Institute of Mathematical Sciences, HBNI, Chennai, India

Meirav Zehavi
Ben-Gurion University of the Negev, Beer-Sheva, Israel

Abstract

We present an algorithm for a class of n-fold ILPs whose existing algorithms in literature are often either (1) based on the *augmentation framework* where one starts with an arbitrary solution and then iteratively moves towards an optimal solution by solving appropriate programs; or (2) require solving a linear relaxation of the program; or (3) are based on decomposition/proximity based arguments. Combinatorial n-fold ILPs is a class of n-fold ILPs introduced and studied by Knop et al. [MP2020] that captures several other problems in a variety of domains. We present a simple and direct algorithm that solves combinatorial n-fold ILPs with unbounded non-negative variables via an application of the Steinitz lemma. Depending on the structure of the input ILP, we also improve upon the existing algorithms in the literature in terms of the running time, thereby showing an improvement that mirrors the one shown by Rohwedder [ICALP2025] contemporaneously and independently.

2012 ACM Subject Classification Theory of computation → Integer programming

Keywords and phrases n-fold integer linear program, parameterized algorithms

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.14

Acknowledgements We thank the reviewers of an earlier draft as well as this for their contribution in improving the clarity and presentation of our results.

1 Introduction

The power of the *linear algebra method* has a long and storied history in the design and analysis of algorithms, [3, 32]. The question of reordering vectors so that all partial sums are bounded was posed by mathematicians Paul Lévy and George Riemann in the early 20th century. The question was first affirmatively answered by Ernst Steinitz [33, 15] showing that such a reordering of vectors, that sum to zero and are all in a unit ball with respect to any arbitrary norm, is indeed possible where all the partial sums can be bounded by a constant that depends only on d, the dimension of the vectors. This result, now eponymously known as the Steinitz Lemma, stated in Proposition 2, has become a cornerstone in our understanding of the geometry of numbers and has led to numerous algorithmic applications centered around problems that have an integer linear program (ILP) formulation [12]. In this paper, we study a simple Steinitz Lemma based non-augmenting algorithm for a class of combinatorial n-fold ILPs.

The usefulness of Steinitz Lemma in modern algorithmic research is perhaps best underscored by its applicability to analyze integer linear programs (ILP), as exhibited by Eisenbrand and Weismantel [12] who improved upon the longstanding best bound of Pa-

© Sushmita Gupta, Pallavi Jain, Sanjay Seetharaman, and Meirav Zehavi;
licensed under Creative Commons License CC-BY 4.0

padimitriou [29]. Papadimitriou's seminal work gave a simpler proof of ILPs being in NP and the first pseudo-polynomial algorithm when the number of constraints is constant. This can be viewed as a *pseudo-parameterized algorithm* with respect to the number of constraints. Subsequently, Jansen and Rohwedder [20] further improved the running time by applying the Steinitz Lemma in an alternative way. In this paper, we are continuing this line of investigation by employing Steinitz Lemma on a special class of ILPs, namely combinatorial n-fold ILP, for the purpose of obtaining better bounds.

Notwithstanding these exciting possibilities, the bottom line is that solving an ILP is known to be NP-complete and that poses a computational challenge when we desire integral solutions only. While we know of certain classes of ILPs, such as *totally unimodular*, *bimodular* [2], *binet* [1] that can be solved in polynomial time, the task of mapping the ILP landscape into families that are tractable under modest restrictions is ongoing. In particular, in recent times researchers are motivated to design parameterized algorithms because they behave like polynomial-time algorithms when the parameter is constant-valued. An algorithm is said to be *fixed-parameter tractable* (FPT) with respect to parameters $k_1, \ldots, k_\ell$, if its running time is of the form $f(k_1, \ldots, k_\ell) \cdot |\text{input}|^{\mathcal{O}(1)}$ for some computable function f. Steinitz Lemma has proven to be useful even in this direction of research, as exhibited in its usefulness in the study of *block-structured* ILPs.

Block-structured ILP. A class of block structured programs called the *n-fold* ILP has come to sharp focus due to their relevance to efficiently solving problems in scheduling [23], fair division [28], and social choice [26] to name a few. For example, in the *high-multiplicity* scheduling setting, the input is succinctly encoded by partitioning the jobs (machines) into *types* such that all jobs (machines) within a type are identical. Note that the number of job (machine) types can be much smaller than the number of jobs (machines). Using n-fold ILP, Knop and Koutecký [23] obtain algorithms for some scheduling problems that are FPT with respect to the number of job (machine) types.

More broadly, the n-fold setting has unlocked the door towards parameterized and approximation algorithms and their rich tool-kit. Formally, we define n-fold ILP as the following

$$\min \left\{ c^\intercal x \mid \mathcal{A}x = b\,, \ell \leq x \leq u\,, x \in \mathbb{Z}^{nt} \right\}, \tag{$\mathbf{P_0}$}$$

where vectors $c \in \mathbb{Z}^{nt}, b \in \mathbb{Z}^{r+ns}, \ell, u \in (\mathbb{Z} \cup \{-\infty, \infty\})^{nt}$ and the constraint matrix

$$\mathcal{A} := \begin{pmatrix} T & T & \cdots & T \\ D & 0 & \cdots & 0 \\ 0 & D & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & D \end{pmatrix}, \text{ such that}$$

the top block $T \in \mathbb{Z}^{r \times t}$ and the diagonal block $D \in \mathbb{Z}^{s \times t}$. Let Δ_X denote an upperbound on the absolute value of an entry in a matrix X, and let Δ_I denote the same for the numbers in the input. Hemmecke et al. [16] were the first to give an FPT algorithm for ($\mathbf{P_0}$) with respect to the parameters $\Delta_\mathcal{A}, r, s, t$. This algorithm runs in time $\mathcal{O}(n^3 t^3 \cdot \Delta_\mathcal{A}^{\mathcal{O}(t(rs+st))} \log(\Delta_c))$. Hence, this has better performance than [29] and [12] that work for general ILPs. Subsequently, Eisenbrand et al. [9], using the Steinitz Lemma in a manner similar to [12], improved the result of [16], in terms of the dependence on t given that $t \geq r, s$. Their algorithm has running time $n^2 t^2 \log \Delta_I \log^2 nt \cdot (rs\Delta_\mathcal{A})^{\mathcal{O}(r^2 s + rs^2)} + \mathbf{LP}$, where $\mathbf{LP}$ denotes the time

needed to solve the LP relaxation of ($\mathbf{P_0}$). This is an FPT algorithm with respect to the parameters $r, s, \Delta_{\mathcal{A}}$. We are naturally interested in algorithms with smaller dependence on t, the number of columns, because t can be as large as $\Delta_{\mathcal{A}}^{r+s}$, a common occurrence in applications involving configuration IPs, as noted in [9]. Koutecky et al. [27] showed an algorithm that runs in time $(nt)^6 \log(nt) \cdot (rs\Delta_{\mathcal{A}})^{\mathcal{O}(r^2s+rs^2)} + \mathbf{LP}$. This is the first strongly polynomial time algorithm for fixed values of $r, s, \Delta_{\mathcal{A}}$. Subsequently, Eisenbrand et al. [10], Jansen et al. [19], and Cslovjecsek et al. [6] have developed algorithms that run in time $(nt)^2 \log^3(nt)(rs\Delta_{\mathcal{A}})^{\mathcal{O}(r^2s+rs^2)} + \mathbf{LP}$, $nt \log^{\mathcal{O}(1)}(nt) \cdot \log(\Delta_I^2) \cdot (rs\Delta_{\mathcal{A}})^{\mathcal{O}(r^2s+s^2)}$, and $(nt)^{1+o(1)} \cdot 2^{\mathcal{O}(rs^2)}(rs\Delta_{\mathcal{A}})^{\mathcal{O}(r^2s+s^2)}$, respectively.

A special class of n-fold ILP known as the *combinatorial n-fold ILP*–where the diagonal block $D = (1, \ldots, 1) \in \mathbb{Z}^{1\times t}$–was introduced by Knop et al. [25] and it captures many problems in various domains including computational social choice, stringology, and scheduling. They showed that such programs can be solved in time $t^{\mathcal{O}(r)}(\Delta_{\mathcal{A}} r)^{\mathcal{O}(r^2)}\mathcal{O}(n^3 \cdot \langle I \rangle) + \mathbf{LP}$, where $\langle I \rangle$ denotes the size of encoding $\langle b, \ell, u, c \rangle$. We would like to note that the result of [6] improves upon this and gives a $(r\Delta_{\mathcal{A}})^{\mathcal{O}(r^2)}(nt)^{1+o(1)}$ algorithm for this special class.

In this paper, we solve a variant of combinatorial n-fold ILP where the variables are non-negative and unbounded from above. However, our constraint matrix is more general because in our setting the top blocks T need not be identical. We formally define our problem and result in Section 1.1. We believe our methodology is conceptually simpler than the earlier approaches, described in the next sub-section, for solving this problem.

1.1 Background and our contribution

In this article we present a simple non-iterative algorithm that leverages the power of the Steinitz Lemma to solve a class of combinatorial n-fold ILP, ($\mathbf{P_1}$). In other words, our approach solves this problem directly and not via a sequence of *augmentation* steps involving the *Graver bases* as exhibited in earlier papers such as [12, 10], or via non-iterative approaches that involve proximity-based arguments [6] or decomposition-based arguments [7].

Formally, we define our setting as follows.

Our setting. We study ILPs of the following form

$$\min \left\{ c^{\intercal} x \mid \mathcal{A}x = b\,, x \in \mathbb{Z}^{nt}_{\geq 0} \right\} \tag{$\mathbf{P_1}$}$$

where vectors $b \in \mathbb{Z}^{r+n}$ and $c \in \mathbb{Z}^{nt\times 1}$; and the constraint matrix

$$\mathcal{A} := \begin{pmatrix} T^{(1)} & T^{(2)} & \cdots & T^{(n)} \\ D & 0 & \cdots & 0 \\ 0 & D & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & D \end{pmatrix}, \text{ such that}$$

the *diagonal* block $D = (1, \ldots, 1) \in \mathbb{Z}^{1\times t}$ and the *top* blocks $T^{(1)}, \ldots, T^{(n)} \in \mathbb{Z}^{r\times t}$.

The main technical contribution of this paper is to prove the following.

▶ **Theorem 1.** *The* ILP ($\mathbf{P_1}$) *can be solved in time* $\mathcal{O}(nt \cdot (n\Delta_{\mathcal{A}}(n+1+4r))^r \cdot q)$, *where* q *denotes the sum of the last* n *entries in* b.

While the expression of time complexity in Theorem 1 is not FPT with respect to $r, \Delta_{\mathcal{A}}$, we note, however, that it allows us to identify a class of instances where our algorithm will outperform the existing algorithms for n-fold ILP, including [10, 19, 6], as exhibited for

the problems Lobbying and Binary Closest String in Section 5. We consider this to be a conceptual contribution of our approach and note that structure of the b vector, and parameters such as q, are worthy of further investigation. We also present an application of our result on the problem Equitable Coloring.

We note that if we apply the *pre-n fold ILP* framework [25, Corollary 23] to $(\mathbf{P_1})$ we would obtain an algorithm with running time $(tn)^r(\Delta_{\mathcal{A}} r)^{\mathcal{O}(r^2)}\mathcal{O}(n^3 \cdot \langle I \rangle)) + \mathbf{LP}$. There is no linear dependence on q, and there is an $(\Delta_{\mathcal{A}} r)^{\mathcal{O}(r^2)}$ term in the running time.

Next, we state the running time of existing algorithms when applied to $(\mathbf{P_1})$: Eisenbrand et al. [9] $n^2t^2 \log^2 nt \log \Delta_I \cdot (r\Delta_{\mathcal{A}})^{\mathcal{O}(r^2)} + \mathbf{LP}$; Jansen et al. [19] $nt \log^{\mathcal{O}(1)}(nt) \cdot \log^2 \Delta_I \cdot (r\Delta_{\mathcal{A}})^{\mathcal{O}(r^2)}$; Cslovjecsek et al. [6] $(nt)^{1+o(1)}2^{\mathcal{O}(r)}(r\Delta_{\mathcal{A}})^{\mathcal{O}(r^2)}$. The algorithms in [16, 27, 7] require the top row block to contain identical matrices and it is not clear how to obtain a solution to our problem using them.

Contemporaneous related works. In [30] (recently accepted in ICALP'25), Rohwedder presented an algorithm for Multi-choice Integer Programming, a problem very closely related to $(\mathbf{P_1})$ (either can be reduced to the other in polynomial time). The running times are identical (in terms of the dependence on parameters), and the techniques are very similar. The main technical difference is in the first phase of the algorithm (Lemma 4) where they present a "time-based" approach instead of the "greedy imbalance-based" approach we present. The main result of that paper is an algorithm for scheduling on uniform machines, which uses the solution for the above problem as a subroutine. Our main contribution is a more detailed presentation of the algorithm, including in Section 4 how we deal with inequalities in the constraints and apply it on Lobbying and Binary Closest String. Moreover, our result can be applied to obtain the corollaries and applications indicated by Rohwedder [30]. We note that dealing with inequalities is an important step in our analysis because we are using the n-fold ILP framework while Rohwedder is not.

We also note that Jansen et al. [17] present a divide-and-conquer-based algorithm (based on the approach in [21]) for a problem that is closely related to the feasibility version of $(\mathbf{P_1})$. It is not clear whether similar guarantees as ours could be achieved by their result, we refer to [30] for a discussion on this. We note that in the updated version [18] of the work (also accepted in IPEC'25), they present an algorithm for $(\mathbf{P_1})$ with running time that has an identical exponential dependence on $n, r, \Delta_{\mathcal{A}}$, but has only a logarithmic dependence on q. These articles are contemporaneous and independent.

Preliminaries

The main technical tool in our result is the Steinitz Lemma, stated below.

▶ **Proposition 2** (Steinitz Lemma, [33, 15]). *Let $\|\cdot\|$ be an arbitrary norm of $\mathbb{R}^d$. Let $x_1, \ldots, x_m \in \mathbb{R}^d$ such that $\sum_{i\in[m]} x_i = 0$ and $\|x_i\| \leq 1$ for each $i \in [m]$. There exists a permutation $\pi \in S_m$ such that all partial sums satisfy*

$$\|\sum_{j\in[k]} x_{\pi(j)}\| \leq d \text{ for each } k \in [m].$$

Eisenbrand and Weismantel [12] generalize the above lemma and show the following.

▶ **Proposition 3** ([12]). *Let $x_1, \ldots, x_m \in \mathbb{R}^d$ such that $\sum_{i\in[m]} x_i = s$ and $\|x_i\|_\infty \leq \Delta$ for each $i \in [m]$. There exists a permutation $\pi \in S_m$ such that all partial sums satisfy*

$$\|\sum_{j\in[k]} x_{\pi(j)} - \frac{k}{m} \cdot s\|_\infty \leq 2\Delta d \text{ for each } k \in [m].$$

Note on variations of n-fold ILP. $(\mathbf{P_1})$ forms a special case of the n-fold integer programs. The following are some of the studied variations of such programs:

1. The n diagonal blocks, which are $D = (1, \ldots, 1)$ above, can be non-identical $s \times t$ matrices, [9, 24];
2. Moreover, *combinatorial* n-fold IPs usually refers to problems where the constraint matrix is a restricted version of $(\mathbf{P_1})$, with the top block, denoted by $(T^{(1)}, \ldots, T^{(n)})$, being n identical matrices, [25, 7];
3. The objective function may be more general than linear, such as separable convex, [25].

Notation. For the sake of convenience, from now on we use Δ to denote $\Delta_{\mathcal{A}}$. For any $z \in \mathbb{Z}_{\geq 0}$, we denote the set $\{1, \ldots, z\}$ by $[z]$. We subdivide the set of entries in a vector $x \in \mathbb{Z}^{nt}$ into x *bricks* $x^{(1)}, \ldots, x^{(n)}$, where each brick is a vector in $\mathbb{Z}^t$ and $x^\intercal = ((x^{(1)})^\intercal, (x^{(2)})^\intercal, \ldots, (x^{(n)})^\intercal)$. For each $i \in [n], j \in [t]$, we use $x^{(i)}_j$ to denote the entry corresponding to the j^{th} variable in the i^{th} brick of x.

For a matrix M, we use $M[\cdot, w]$ and $M[w, \cdot]$ to denote the w^{th} column and the w^{th} row of M respectively. For a matrix M, we use $\mathrm{psum}(M, j)$ to denote the partial sum up to column j in M: that is, $\mathrm{psum}(M, j) = \sum_{w \in [j]} M[\cdot, w]$. For a matrix M and an entry e, we use $\mathtt{occ}_M(e, j)$ to denote the number of occurrences of e in M up till column j. For a matrix M and a permutation of its columns σ, we use M_σ to denote the matrix where $M_\sigma[\cdot, j] = M[\cdot, \sigma(j)]$.

With the definition of bricks, we have that any feasible solution x satisfies

$$\begin{pmatrix} T^{(1)} & T^{(2)} & \cdots & T^{(n)} \\ 1^T & 0 & \cdots & 0 \\ 0 & 1^T & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1^T \end{pmatrix} \begin{pmatrix} x^{(1)} \\ x^{(2)} \\ \vdots \\ x^{(n)} \end{pmatrix} = \begin{pmatrix} b^{(0)} \\ b^{(1)} \\ b^{(2)} \\ \vdots \\ b^{(n)} \end{pmatrix}.$$

Due to the matrix structure, the problem $(\mathbf{P_1})$ is equivalent to the following question, which we will focus on from now. Let $q = \sum_{i \in [n]} b^{(i)}$. Does there exist $x \in \mathbb{Z}^{nt}_{\geq 0}$ such that

- [all but the top row block] $\sum_{j \in [t]} x^{(i)}_j = b^{(i)}$, for each $i \in [n]$,
- [top row block] there is a sequence of columns, say $v_1, \ldots, v_q$, such that $\sum_{k \in [q]} v_k = b^{(0)}$, and column $T^{(i)}[\cdot, j]$ appears $x^{(i)}_j$ times in the sequence,
- $c^\intercal x$ is maximized?

If there is no such x, then we decide that it is infeasible.

2 Technical Overview

Suppose that the given program $(\mathbf{P_1})$ admits a solution x. From x, we obtain a sequence of vectors $v_1, \ldots, v_q$ such that

1. $q = \sum_{i \in [n]} b^{(i)}$;
2. each v_j is a column of one of the top blocks $T^{(1)}, \ldots, T^{(n)}$;
3. $\sum_{j \in [q]} v_j = b^{(0)}$;
4. the column $T^{(i)}[\cdot, k]$ appears $x^{(i)}_k$ many times in the sequence.

Consider the $1 \times q$ matrix M with entries from $[n]$ where $M[j] = i$ if $v_j = T^{(i)}[\cdot, k]$ for some $k \in [t]$, and each $i \in [n]$ is present exactly $b^{(i)}$ times in M. Informally, through M we identify the block from which each vector is obtained. The first step of our approach is to "balance" this one-row matrix M.

Suppose that the symbol (or element) $e \in [n]$ appears m_e times in M. Consider a random rearrangement of the matrix M. The expected number of occurrences of e in the first j columns, by the linearity of expectation, is $(j/q) \cdot m_e$. This is because the probability that a random rearrangement of symbols in M contains e in a particular index is m_e/q, and the linearity of expectation is applied over the first j columns. Let $\mathtt{occ}_M(e, j)$ denote the number of occurrences of symbol e between columns 1 to j in M. The **imbalance** of a symbol e at a column j in M is defined as

$$\mathtt{imb}_M(e, j) \coloneqq \mathtt{occ}_M(e, j) - (j/q) \cdot m_e. \tag{*}$$

Thus, the imbalance of a symbol e at a particular column is the difference between the number of occurrences of e and the expected number of occurrences of e until that column in a random rearrangement of the entries. We show that we can, in time $\mathcal{O}(nq)$, rearrange the symbols such that the matrix has bounded imbalance[1].

▶ **Lemma 4** (Balancing a one-row matrix). *Let M be a $1 \times q$ matrix with entries from $[n]$. There is a permutation σ such that for each symbol $e \in [n]$ and column $j \in [q]$, $-n \leq \mathtt{imb}_{M_\sigma}(e, j) \leq 1$. Moreover, the matrix M_σ can be computed in time $\mathcal{O}(nq)$.*

For each block $i \in [n]$, we do the following. Consider the $r \times b^{(i)}$ matrix formed by all the $b^{(i)}$-many vectors in the sequence corresponding to the block i, say $V^{(i)}$. By Proposition 3, there is a permutation of the vectors in $V^{(i)}$, say $\pi^{(i)} \in S_{b^{(i)}}$, such that each coordinate of the j^{th} partial sum is at most $2r\Delta$ away from the $(j/b^{(i)})^{th}$ fraction of the total sum of vectors in $V^{(i)}$. We construct a new $r \times q$ matrix O from M_σ by replacing the entries with vectors from those blocks. We place these vectors $V^{(i)}_{\pi^{(i)}}$ (in the same order) in O based on where i occurs in M_σ: the w^{th} vector in $V^{(i)}_{\pi_{(i)}}$ is placed in the w^{th} occurrence of i in M_σ. We show that all partial sums in O are bounded.

▶ **Lemma 5** (Bounding the partial sums). *For each $j \in [q]$ and $k \in [r]$, we have $-n\Delta(n+2r) \leq \mathrm{psum}(O, j)[k] - \frac{j}{q} b^{(0)}[k] \leq n\Delta(1 + 2r)$.*

We construct a weighted directed acyclic graph G based on the bounds we obtained in Lemma 5. The key property of G is that any shortest path between two particular vertices in G corresponds to an optimal solution of the program and vice versa. If there is no such path, then we assert that there is no solution. Otherwise, we compute an optimal solution corresponding to the path. Thus, we have our main result, Theorem 1.

3 Proof of Theorem 1

In this section, we prove Lemma 4 in Section 3.1, followed by Lemma 5 in Section 3.2, and then present the overall algorithm in Section 3.3 which forms a proof of Theorem 1.

[1] We would like to note that a similar result can be obtained by applying the Steinitz Lemma on the $q * n$ matrix M' where for each $j \in [q]$, $M'[\cdot, j]$ is a vector containing 1 in position $M[j]$ and 0 in all other positions. However, such an application will result in a weaker imbalance upper-bound of n and a slightly more complicated algorithm.

3.1 Balancing a one-row matrix: Proof of Lemma 4

We restate Lemma 4 for convenience.

▶ **Lemma 4** (Balancing a one-row matrix). *Let M be a $1 \times q$ matrix with entries from $[n]$. There is a permutation σ such that for each symbol $e \in [n]$ and column $j \in [q]$, $-n \leq \mathtt{imb}_{M_\sigma}(e, j) \leq 1$. Moreover, the matrix M_σ can be computed in time $\mathcal{O}(nq)$.*

Algorithm 1 An algorithm to balance a one-row matrix.

Input: the number of occurrences of each entry in the $1 \times q$ matrix M

1: $M_\sigma \leftarrow$ an empty $1 \times q$ matrix
2: **for** j from 1 to q **do**
3: Let $y \leftarrow \arg\min_{e \in [n]} \mathtt{imb}_{M_\sigma}(e, j)$
4: Set $M_\sigma[j] \leftarrow y$
5: **end for**
6: **return** M_σ

Consider M_σ, the $1 \times q$ matrix with entries in $[n]$, constructed by Algorithm 1. The algorithm starts with an empty M_σ; then, it iteratively fills the symbols column-wise by finding the symbol with minimum imbalance (note that imbalances can be negative).

Maintaining the imbalances of each of the n distinct symbols and finding a symbol with minimum imbalance (Line 3) can be done in time $\mathcal{O}(n)$. This can be done by storing and updating the imbalances of each of the n symbols. Overall, Algorithm 1 runs in time $\sum_{j \in [q]} \mathcal{O}(n) = \mathcal{O}(nq)$.

Next, we show the bounds on the symbol imbalances in M_σ. Recall that a symbol $e \in [n]$ appears m_e times in M. Fix any $j \in [q]$. Consider the stage of the algorithm where the first j columns of M_σ are filled. Now we show the imbalance upper-bound in the lemma. Observe that the quantity $\mathtt{imb}_{M_\sigma}(e, j)$, when viewed as a sequence indexed by j, increases only when symbol e is in the entry $M_\sigma[j]$. Assume for the sake of contradiction that $\mathtt{imb}_{M_\sigma}(e', j') > 1$ for some $e' \in [n]$ and $j' \in [q]$. Consider the first index j such that $\mathtt{imb}_{M_\sigma}(e', j) > 1$ and $M_\sigma[j] = e'$. Due to the greedy choice in the construction of M_σ, we have that $\mathtt{imb}_{M_\sigma}(e, j) > 0$ for each $e \in [n]$. We then arrive at a contradiction since

$$j = \sum_{e \in [n]} \mathtt{occ}_{M_\sigma}(e, j) = \sum_{e \in [n]} \left(\mathtt{imb}_{M_\sigma}(e, j) + \frac{j}{q} m_e \right) > \sum_{e \in [n]} \left(\frac{j}{q} m_e \right) = j.$$

Next, we show the imbalance lower-bound in the lemma. A crucial property that we will use to show the bound is that the sum of imbalances at column j is zero.

▷ Claim 5.1.

$$\sum_{e \in [n]} \mathtt{imb}_{M_\sigma}(e, j) = 0.$$

Proof.

$$\sum_{e \in [n]} \mathtt{imb}_{M_\sigma}(e, j) = \sum_{e \in [n]} \left(\mathtt{occ}_{M_\sigma}(e, j) - \frac{j}{q} m_e \right) = \sum_{e \in [n]} \mathtt{occ}_{M_\sigma}(e, j) - \sum_{e \in [n]} \frac{j}{q} m_e = j - \frac{j}{q} q = 0. \quad \triangleleft$$

Assume for contradiction that $\mathtt{imb}_{M_\sigma}(e', j') < -n$ for some $e' \in [n]$ and $j' \in [q]$. The sum of symbol imbalances at j' is

$$\sum_{e \in [n]} \mathtt{imb}_{M_\sigma}(e, j') = \mathtt{imb}_{M_\sigma}(e', j') + \sum_{e \in [n] \setminus \{e'\}} \mathtt{imb}_{M_\sigma}(e, j') < -n + \sum_{e \in [n] \setminus \{e'\}} \mathtt{imb}_{M_\sigma}(e, j').$$

Combining the above with Claim 5.1, there exists an element $e \in [n] \setminus \{e'\}$ such that $\texttt{imb}_{M_\sigma}(e, j') > 1$ and thus we have a contradiction.

Combining the lower and upper-bounds, we infer that $-n \leq \texttt{imb}_{M_\sigma}(e, j) \leq 1$ for each $e \in [n]$ and $j \in [q]$. What is left to be shown is that the matrix M_σ is a permutation of the columns of M. For that, it is sufficient to prove that for each $e \in [n]$, the number of occurrences of element e in M_σ is m_e. Assume for contradiction that M_σ is not a permutation of columns of M. We observe that since $\sum_{e\in[n]} \texttt{occ}_{M_\sigma}(e, q) = q = \sum_{e\in[n]} m_e$, there exists a symbol $e' \in [n]$ such that $\texttt{occ}_{M_\sigma}(e', q) > m_{e'}$. Consider the last index j such that the j^{th} column in M_σ contains e'. We have

$$\texttt{imb}_{M_\sigma}(e', j) = \texttt{occ}_{M_\sigma}(e', j) - \frac{j}{q} m_{e'} = \texttt{occ}_{M_\sigma}(e', q) - \frac{j}{q} m_{e'} \geq m_{e'} + 1 - \frac{j}{q} m_{e'}.$$

If $j < q$, then $\texttt{imb}_{M_\sigma}(e', j) > 1$ and we have a contradiction to the upper-bound in the lemma. Otherwise, we have $j = q$. Observe that $\texttt{imb}_{M_\sigma}(e', q) = 1$. By Line 3, we have that $\texttt{imb}_{M_\sigma}(e, q) \geq 0$ for each $e \in [n] \setminus \{e'\}$. However, this implies that $\sum_{e\in[n]} \texttt{imb}_{M_\sigma}(e, q) \geq 1$, a contradiction to Claim 5.1. This completes the proof of the lemma.

3.2 Bounding the partial sums: Proof of Lemma 5

We restate Lemma 5 for convenience.

▶ **Lemma 5** (Bounding the partial sums). *For each $j \in [q]$ and $k \in [r]$, we have $-n\Delta(n+2r) \leq \text{psum}(O, j)[k] - \frac{j}{q} b^{(0)}[k] \leq n\Delta(1 + 2r)$.*

Before we prove Lemma 5, we recall the construction of O and relevant notation that were introduced in Section 2. We consider an optimal solution x of $(\mathbf{P_1})$ and from it obtain a sequence of vectors $v_1, \ldots, v_q$ satisfying the conditions given in Section 2. Next, using Algorithm 1 we compute M_σ, a $1 \times q$ matrix such that for each symbol $i \in [n]$

1. i occurs $b^{(i)}$ times in M_σ;
2. for each $j \in [q]$, $-n \leq \texttt{imb}_{M_\sigma}(i, j) \leq 1$. (i.e., $-n + (j/q)b^{(i)} \leq \texttt{occ}_{M_\sigma}(i, j) \leq 1 + (j/q)b^{(i)}$).

For each block $i \in [n]$, we do the following. Consider $V^{(i)}$, the $r \times b^{(i)}$ matrix formed by the set of columns in the sequence corresponding to block i. Applying Proposition 3 on $V^{(i)}$, we obtain a permutation $\pi_i \in S_{b^{(i)}}$ such that all partial sums of $V^{(i)}_{\pi_i}$ are bounded. In particular,

$$\|\text{psum}(V^{(i)}_{\pi_i}, w) - \frac{w}{b^{(i)}} \cdot \text{tot}(V^{(i)})\|_\infty \leq 2r\Delta \text{ for all } w \in [b^{(i)}], \tag{1}$$

where $\text{tot}(V^{(i)})$ is the sum of the columns of $V^{(i)}$ (note that $\text{tot}(V^{(i)}) = \text{tot}(V^{(i)}_{\pi_i})$).

Algorithm 2 An algorithm to construct the matrix O.

```
1: O ← an empty r × q matrix
2: w_1, ..., w_n ← 0
3: for j from 1 to q do
4:     e ← M_σ[j]
5:     w_e ← w_e + 1
6:     O[·, j] ← V^(i)_{π_i}[·, w_e]
7: end for
8: return O
```

Next, we construct O, an $r \times q$ matrix as described in Algorithm 2. Essentially, we place the columns of $V^{(i)}_{\pi_i}$ on the positions where i appears in M_σ. Now we are ready to show that this matrix has bounded partial sums.

For any $j \in [q]$ and $k \in [r]$, we bound the partial sum of O at index k of column j as

$$\begin{aligned}
\mathrm{psum}(O,j)[k] &= \sum_{i\in[n]} \mathrm{psum}(V^{(i)}_{\pi_i}, \mathtt{occ}_{M_\sigma}(i,j))[k] \leq \sum_{i\in[n]} \left(\frac{\mathrm{occ}(i,j)}{b^{(i)}} \mathrm{tot}(V^{(i)}_{\pi_i})[k] + 2r\Delta \right) \\
&\leq \sum_{i\in[n]} \left(\frac{1+\frac{j}{q}b^{(i)}}{b^{(i)}} \mathrm{tot}(V^{(i)})[k] + 2r\Delta \right) \\
&= \sum_{i\in[n]} \left(\frac{1}{b^{(i)}} \mathrm{tot}(V^{(i)})[k] + \frac{j}{q} \mathrm{tot}(V^{(i)})[k] + 2r\Delta \right) \\
&\leq \sum_{i\in[n]} \left(\frac{1}{b^{(i)}} \Delta b^{(i)} + \frac{j}{q} \mathrm{tot}(V^{(i)})[k] + 2r\Delta \right) = \frac{j}{q} b^{(0)}[k] + \sum_{i\in[n]} (\Delta + 2r\Delta) \\
&= \frac{j}{q} b^{(0)}[k] + n\Delta(1+2r).
\end{aligned}$$

Similarly, we have the following lower bound.

$$\begin{aligned}
\mathrm{psum}(O,j)[k] &= \sum_{i\in[n]} \mathrm{psum}(V^{(i)}_{\pi_i}, \mathtt{occ}_{M_\sigma}(i,j))[k] \geq \sum_{i\in[n]} \left(\frac{\mathrm{occ}(i,j)}{b^{(i)}} \mathrm{tot}(V^{(i)}_{\pi_i})[k] - 2r\Delta \right) \\
&\geq \sum_{i\in[n]} \left(\frac{-n+\frac{j}{q}b^{(i)}}{b^{(i)}} \mathrm{tot}(V^{(i)})[k] - 2r\Delta \right) \\
&= \sum_{i\in[n]} \left(\frac{-n}{b^{(i)}} \mathrm{tot}(V^{(i)})[k] + \frac{j}{q} \mathrm{tot}(V^{(i)})[k] - 2r\Delta \right) \\
&\geq \sum_{i\in[n]} \left(\frac{-n}{b^{(i)}} \Delta b^{(i)} + \frac{j}{q} \mathrm{tot}(V^{(i)})[k] - 2r\Delta \right) = \frac{j}{q} b^{(0)}[k] + \sum_{i\in[n]} (-n\Delta - 2r\Delta) \\
&= \frac{j}{q} b^{(0)}[k] - n\Delta(n+2r).
\end{aligned}$$

3.3 The algorithm

Algorithm 3 An algorithm to solve $(\mathbf{P_1})$.

1: Compute M_σ using Algorithm 1.
2: Construct the graph $G = (V, A)$ as given in Construction 1.
3: Compute a shortest path P between $h_{0,0}$ and $h_{q,b^{(0)}}$.
4: Return an optimal solution corresponding to P.

We now describe a direct algorithm to solve $(\mathbf{P_1})$. From now on, we suppose that the given program admits an optimal solution x^*. Otherwise, the algorithm asserts that there is no solution for the program and we are correct. Firstly, using Algorithm 1, we compute M_σ, a $1 \times q$ matrix that satisfies the conditions of Lemma 4. Note that the existence of x^* alone is sufficient to compute M_σ since Algorithm 1 only requires the total number of occurrences of each symbol (which is $b^{(i)}$ for each $i \in [n]$).

The existence of x^* now implies that there is a matrix O which satisfies the conditions of Lemma 5. To find such an x^*, we find the existence of a matrix satisfying Lemma 5, by finding shortest paths in certain graphs - in a way similar to [12]. We construct a weighted directed acyclic graph $G = (V, A)$ as follows.

▶ Construction 1.

1. (Vertices) For each $j \in [n]$ and each integer vector $v \in \mathbb{Z}^r$ such that

$$(j/q)b^0[k] - n\Delta(n+2r) \leq v[k] \leq (j/q)b^0[k] + n\Delta(1+2r)$$

for each $k \in [r]$, we add a vertex $h_{j,v}$ to V. Lastly, we add the vertex $h_{0,0}$ to V.

2. (Edges) For each $j \in [q]$, we do the following. Suppose that $M_\sigma[j] = i$. We add the arc $(h_{v_1,j-1}, h_{v_2,j})$ to A if $v_2 - v_1$ is the column $T^{(i)}[\cdot, k]$ for some $k \in [t]$, and the weight of the arc is set as $c_k^{(i)}$.

In the following claim, we bound the sizes of $|V|$ and $|A|$.

▷ Claim 6. $|V| \leq 1 + q \cdot (n\Delta(n+1+4r))^r$ and $|A| = \mathcal{O}(nt|V|)$. Moreover, we can construct G in time $\mathcal{O}(qnt \cdot (n\Delta(n+1+4r))^r)$.

Proof. For each $j \in [q]$, the number of integer vectors v such that $(j/q)b^0[k] - n\Delta(n+2r) \leq v[k] \leq (j/q)b^0[k] + n\Delta(1+2r)$ for each $k \in [r]$ is at most $(n\Delta(n+1+4r))^r$. This is because for a fixed $k \in [r]$, the number of possible values that $v[k]$ can take is $n\Delta(n+1+4r)$. Summing over all $j \in [q]$ and then counting the vertex $h_{0,0}$, we have the claim. Note that we can construct V in time proportional to its size.

To construct A, we can iterate over all vertices $h_{j,v}$, and in time nt compute all arcs incident on $h_{j,v}$. Thus, we have $|A| = \mathcal{O}(nt|V|)$, and we can construct the graph G in time $\mathcal{O}(|V| + nt|V|) = \mathcal{O}(qnt \cdot (n\Delta(n+1+4r))^r)$. ◁

By the definition of G and the existence of x^*, we have that there is a path of cost $c^\intercal x^*$ in G between vertices $h_{0,0}$ and $h_{b^{(0)},q}$ with cost $c^\intercal x^*$ and this is the shortest such path. We can compute such a path by using BFS in time $\mathcal{O}(|A|) = \mathcal{O}(qnt \cdot (n\Delta(n+1+4r))^r)$. To compute an optimal solution for $(\mathbf{P_1})$ corresponding to P, we simply trace the path and set the variables according to the vertices that appear on the path.

4 Dealing with inequalities in constraints

In this section, we generalize our main result by showing that programs which contain "certain" inequalities in the constraints can be solved in the same time that we take for $(\mathbf{P_1})$ where we have $\mathcal{A}x = b$. This is towards the ease of expressing problems involving inequalities as n-fold integer programs which we will see later in Section 5.

We call the upper rows $(T^{(1)}\ T^{(2)}\ \cdots\ T^{(n)})x = b^{(0)}$ as *globally uniform constraints*, and the lower rows $1^\intercal x^{(i)} = b^{(i)}$ as *locally uniform constraints*. Consider the generalization of $(\mathbf{P_1})$

1. where the globally uniform constraints may contain either of $\{\leq, =, \geq\}$;
2. where the locally uniform constraints may contain either of $\{\leq, =\}$.

Let $A = (1, \ldots, 1) \in \mathbb{Z}^{1\times t}$, $T^{(1)}, \ldots, T^{(n)} \in \mathbb{Z}^{r\times t}$, $b \in \mathbb{Z}^{r+n}$, and $c \in \mathbb{Z}^{nt\times 1}$. Formally, we consider the following generalization.

$$\min\left\{c^\intercal x \mid \mathcal{A}x \diamond b, x \in \mathbb{Z}^{nt}_{\geq 0}\right\} \qquad (\mathbf{P_2})$$

$$\text{where } \mathcal{A} := \begin{pmatrix} T^{(1)} & T^{(2)} & \cdots & T^{(n)} \\ A & 0 & \cdots & 0 \\ 0 & A & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A \end{pmatrix}$$

and $\diamond$ is a sequence of inequalities satisfying conditions 1 and 2.

We emphasize that the generalization does not allow locally uniform constraints to contain "$\geq$". We would like to note that our approach at a high level is almost the same as that of [25], and the main difference lies in the fact that (unlike as in their model,) here the variables in the programs we consider are non-negative and have no explicit lower or upper bounds. The proof strategy is to construct an appropriate program of the form $(\mathbf{P_1})$ by adding new (slack) variables. Let $\psi \coloneqq 2 + 2\|c\|_\infty \cdot \max_{i\in[n]}\{b^{(i)}\}$. Observe that $\psi/2$ is strictly larger than the maximum value the objective function in $(\mathbf{P_2})$ can take.

First, we deal with the locally uniform constraints in $(\mathbf{P_2})$. We add n variables: for each $i \in [n]$, we add $x^{(i)}_{t+1}$ and replace $T^{(i)}$ by $(T^{(i)}\ \mathbf{0})$ where $\mathbf{0}$ is the $r \times 1$ matrix (i.e column) containing zeros. If the i^{th} constraint has a "$\leq$", then we set $c^{(i)}_{t+1} = 0$. Otherwise it contains an "=", and we set $c^{(i)}_{t+1} = \psi$.

Next, we deal with the globally uniform constraints in $(\mathbf{P_2})$. Without loss of generality, we assume that each constraint either contains a "$\leq$" or an "=". This is a safe assumption because if there is a constraint containing a "$\geq$", then we can negate that constraint (which flips the inequality in that constraint to "$\leq$") and obtain a program that is still of the form we consider. We replace each $(T^{(i)}\ \mathbf{0})$ by $(T^{(i)}\ \mathbf{0}\ I_r)$, where I_r is the $r \times r$ identity matrix. We also introduce a new $(n+1)^{st}$ block $(Z_r\ \mathbf{0}\ I_r)$ where Z_r is the $r \times t$ matrix consisting of zeros, and $\mathbf{0}$ is the $r \times 1$ matrix consisting of zeros. The coefficients of the new variables in the first n blocks are set to ψ. That is, for each $i \in [n]$ and $j \in [t+2, t+r+1]$, we set $c^{(i)}_j = \psi$. For the $(n+1)^{st}$ block, we set the coefficients as follows. For each $j \in [t+r+1]$, we set $c^{(n+1)}_j = 0$. Then, we set $b^{(n+1)} = 2r\Delta \cdot \max_{i\in[n]}\{b^{(i)}\}$, where Δ is the largest entry in the constraint matrix $\mathcal{A}$ we started with. This concludes the construction of the program of the form $(\mathbf{P_1})$.

Consider any feasible solution y to the initial program $(\mathbf{P_2})$. We define a solution x to the constructed program of the form $(\mathbf{P_1})$ of the same cost by setting the newly added variables appropriately. Firstly, we set $x^{(i)}_j = y^{(i)}_j$ for each $i \in [n]$ and $j \in [t]$. Recall that we assume w.l.o.g. that each globally uniform constraint contains either a "$\leq$" or an "=". For each $k \in [r]$, we deal with the k^{th} globally uniform constraint in $(\mathbf{P_2})$ as follows. We set $x^{(n+1)}_{t+1+k} = b^{(0)}[k] - \mathcal{A}[k,\cdot]y$. Observe that $x^{(n+1)}_{t+1+k}$ is precisely the slack in the k^{th} globally uniform constraint corresponding to solution y, and it is a number that is at most $2\Delta \cdot b^{(k)}$. For each $i \in [n]$, we deal with the i^{th} locally uniform constraint in $(\mathbf{P_2})$ as follows. We set $x^{(i)}_{t+1} = b^{(i)} - (1\ldots1)^\intercal y^{(i)}$. Finally, we set $x^{(n+1)}_{t+1} = b^{(n+1)} - \sum_{k\in[r]} x^{(n+1)}_{t+1+k}$. Similar to before, observe that $x^{(i)}_{t+1}$ is precisely the slack in the i^{th} locally uniform constraint corresponding to solution y, and it is a number that is at most $b^{(i)}$. All other variables whose values have not yet been set in x, we set them to 0.

We claim that x is a solution to the constructed program $(\mathbf{P_1})$ of the same cost. Firstly, it is a feasible solution by construction. Secondly, the cost is the same because any newly added variable that is set to a non-zero value corresponds to a "$\leq$" constraint, and its coefficient is 0 in $(\mathbf{P_1})$. For the other direction, suppose that we are given a solution x to $(\mathbf{P_1})$. If the cost of x is at least $\psi/2$ (which is strictly larger than the value of any feasible solution in $(\mathbf{P_2})$), then we can assert that $(\mathbf{P_2})$ does not admit any solution. Otherwise, we obtain a solution to $(\mathbf{P_2})$ of the same cost by discarding the newly added variables.

Now that we have established how to compute the solution of one program given that of the other, we move on to assess the time to construct $(\mathbf{P_1})$ and the change in parameters. Each locally uniform constraint can be dealt with separately in time $\mathcal{O}(nt)$ by adding a column corresponding to it and updating c appropriately. All globally uniform constraints can be dealt with together in time $\mathcal{O}(n \cdot rnt)$ by adding r columns to each block, adding a new block, and then updating b appropriately. In total, we can construct $(\mathbf{P_1})$ in time $\mathcal{O}(n^2rt)$. Thus, given $(\mathbf{P_2})$, we can construct $(\mathbf{P_1})$ in time $\mathcal{O}(n^2rt)$.

Next, we assess the change in parameters: the number of blocks is $n+1$, the number of columns in a block is $t+r+1$, the number of globally uniform constraints is r, the parameter q (which is the sum of the right hand sides of the locally uniform constraints) becomes $q + 2r\Delta \max_{i\in[n]}\{b^{(i)}\}$, and the largest entry Δ remains the same since the newly added entries to the matrix are either 0 or 1. Applying Theorem 1, we can solve $(\mathbf{P_1})$ in time

$$\mathcal{O}((q + 2r\Delta \max_{i\in[n]}\{b^{(i)}\})(n+1)(t+r+1)\cdot((n+1)\Delta(n+1+1+4r))^r) \tag{2}$$

$$\leq \mathcal{O}(qr\Delta(n+1)(t+r+1)\cdot((n+1)\Delta(n+2+4r))^r) \tag{3}$$

$$\leq \mathcal{O}(qrt\cdot((n+1)\Delta(n+2+4r))^{r+1}). \tag{4}$$

Summing with the time to construct $(\mathbf{P_1})$, we obtain the total time to solve $(\mathbf{P_2})$.

▶ **Corollary 7.** *The ILP* $(\mathbf{P_2})$ *can be solved in time* $\mathcal{O}(qrt\cdot((n+1)\Delta(n+2+4r))^{r+1})$.

5 Applications

In this section, we apply our main result on three problems: Lobbying, Binary Closest String, and Equitable Coloring.

5.1 Lobbying in multiple referenda

Given the binary approvals (equivalently, yes or no answers) of n voters on m issues, the question of Lobbying is whether the lobby can choose k voters to be "influenced" so that each issue gets a majority of approvals. By influence a voter, we mean gaining complete control over the voter, and getting her approval on all issues. Christian et al. [5] introduced Lobbying and modeled it as the following binary matrix modification problem.

> Lobbying
> **Input:** A matrix $A \in \{0,1\}^{w\times m}$ and an integer $k \geq 0$
> **Question:** Can one choose k rows and flip all 0s to 1s in these rows so that in the resulting matrix every column has more 1s than 0s?

First, we recall how Bredereck et al. [4] express Lobbying as an integer program. Let ℓ denote the number of distinct types of rows in M: two rows are said to be of the same type if they are identical. Observe that $\ell \leq 2^m$. Let $c(r_1),\ldots,c(r_\ell)$ denote the number of rows of each type. For each $i\in[\ell]$ and $j\in[m]$, let $B_j(r_i)=1$ if the j^{th} column of row type r_i has value 0, and let $B_j(r_i)=0$ otherwise. For each $i\in[\ell]$, let b_i be an integer variable that indicates the number of times one has to modify a row of type r_i. For each $j\in[m]$, let g_j denote the number of missing 1s to make column j have more 1s than 0s. Thus, we have the following two constraints: (1) $0\leq b_i\leq c(r_i)$ for each $i\in[\ell]$; (2) $g_j \leq \sum_{i\in[\ell]} b_i\cdot B_j(r_i)$ for each $j\in[m]$. Overall, we have the following program which is of the form $(\mathbf{P_2})$. The objective is to minimize $\sum_{i\in[\ell]} b_i$ subject to

$$\begin{pmatrix} -B_1(r_1) & -B_1(r_2) & \cdots & -B_1(r_\ell) \\ -B_2(r_1) & -B_2(r_2) & \cdots & -B_2(r_\ell) \\ \vdots & \vdots & \ddots & \vdots \\ -B_m(r_1) & -B_m(r_2) & \cdots & -B_m(r_\ell) \\ 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ \\ b_\ell \end{pmatrix} \leq \begin{pmatrix} -g_1 \\ -g_2 \\ \vdots \\ -g_m \\ c(r_1) \\ c(r_2) \\ \vdots \\ \\ c(r_\ell) \end{pmatrix}.$$

Observe that the optimal solution is at most k if and only if there are k rows that can be chosen to satisfy the objective in question. The parameters corresponding to the above program are $r = m$, $t = 1$, $n = \ell \leq 2^m$, $\Delta = 1$, and $q = \sum_{k\in[m]} g_k + \sum_{i\in[\ell]} c(r_\ell) \leq wm + w = (m+1)w$. Given the input, we can construct the above program in time $\mathcal{O}(2^m w)$. Applying Corollary 7, we have the following.

▶ **Theorem 8.** *Lobbying can be solved in time* $2^{\mathcal{O}(m^2)} \cdot w^{\mathcal{O}(1)}$.

The best known dependency on m prior to this work, established by Knop et al. [25], is $m^{\mathcal{O}(m^2)}$. Next, we discuss the consequence of existing algorithms on the above program. Using the algorithms for standard ILPs, including [22, 21], results in algorithms that are double exponential with respect to m. Using the algorithms for n-fold ILP, including [19, 6], results in algorithms that run in time $m^{\mathcal{O}(m^2)} \log w + \mathcal{O}(2^m \cdot w)$ (the $\mathcal{O}(2^m \cdot w)$ term is for the construction of the program). Thus, we establish an improvement in running time in terms of the dependence on m.

5.2 Stringology

We consider δ-Multi Strings, a general problem defined by Knop et al. in [25] that captures many previously studied string problems including Closest String, Farthest String, d-Mismatch, Distinguishing String Selection, Optimal Consensus, and Closest to Most Strings. See [25] for a clear description of how it captures the other problems.

δ-Multi Strings
Input: k strings $s_1, \ldots, s_k$, each of length L from an alphabet $\Sigma \cup \{\star\}$ (where $\star$ denotes a wildcard: a special character that matches with all characters in Σ), distance lower-bounds $d_1, \ldots, d_k \in \mathbb{N}$, and distance upper-bounds $D_1, \ldots, D_k \in \mathbb{N}$, distance function $\delta \colon \Sigma^* \times \Sigma^* \to \mathbb{N}$, and a binary parameter $\beta \in \{0, 1\}$.
Question: Find a string $y \in \Sigma^L$ such that
- y minimizes $\beta \cdot \left(\sum_{h=1}^{k} \delta(y, s_h)\right)$;
- for each s_h, the distance (based on δ) between y and s_h, denoted by $\delta(y, s_h)$, is at least d_h and at most D_h.

A restriction of δ that makes the problem expressible by n-fold integer programs is when δ is a *character-wise wildcard-compatible*: for any two strings $x, y \in (\Sigma \cup \{\star\})^L$, $\delta(x, y) = \sum_{j=1}^{L} \delta(x[j], y[j])$ and $\delta(c, \star) = 0$ for all $c \in \Sigma$. From now on, we consider only this restriction.

We begin by viewing the input as a $k \times L$ character matrix S where the i^{th} row contains the string s_i. We say that two columns in S are of the same type if they are identical (i.e., both in terms of the number of occurrences of each symbol and the order of the occurrences). Since A is a $k \times L$ matrix with entries from $\Sigma \cup \{\star\}$, there are at most $w = (|\Sigma| + 1)^k$ column-types in S. W.l.o.g., we assume that each column-type is denoted by an element in $[w]$. A solution $z \in \Sigma^L$ can be mapped to a $(k+1) \times L$ matrix where the first k rows are the k input strings and the $(k+1)^{st}$ row is z. We construct an integer program that detects the existence of such a matrix.

For each $i \in [w]$ and $c \in \Sigma$, the variable $x_c^{(i)}$ indicates the number of columns in the solution matrix which contain i followed by c. For each $i \in [w]$, let $\mathbf{e_i}$ denote the $k \times 1$ string corresponding to column-type i. The globally uniform constraints are the following. For each $j \in [k]$, we have

$$\sum_{c\in\Sigma} \sum_{i\in[w]} x_c^{(i)} \; \delta(c, \mathbf{e_i}[j]) \geq d_j; \qquad \sum_{c\in\Sigma} \sum_{i\in[w]} x_c^{(i)} \; \delta(c, \mathbf{e_i}[j]) \leq D_j.$$

For each $i \in [w]$, let o_i denote the number of columns of type i; the i^{th} locally uniform constraint constraint is

$$\sum_{c \in \Sigma} x_c^{(i)} = o_i.$$

The objective function is $\beta \cdot \left(\sum_{h=1}^{k} \delta(y, s_h)\right) = \beta \cdot \left(\sum_{h \in [k]} \sum_{i \in [w]} \sum_{c \in \Sigma} x_c^{(i)} \delta(\mathbf{e}_i[h], c)\right)$. Note that it is a linear function on the variables since $\beta \in \{0, 1\}$, and thus we can set c appropriately. The parameters corresponding to the program are $n = (|\Sigma| + 1)^k$, $r = 2k$, $t = |\Sigma|$, $q = \sum_{i \in [w]} o_i = L$, and $\Delta = \max_{c_1, c_2} \delta(c_1, c_2)$. Given the input, we can construct the above program in time $\mathcal{O}((|\Sigma| + 1)^k kL)$. Applying Corollary 7, we have the following.

▶ **Theorem 9.** *δ-Multi Strings can be solved in time $(\Delta k)^{\mathcal{O}(k)}(|\Sigma|)^{\mathcal{O}(k^2)} \cdot L^{\mathcal{O}(1)}$, where $\Delta = \max_{c_1, c_2} \delta(c_1, c_2)$, and δ is a character-wise wildcard-compatible function.*

Knop et al. [25] showed that δ-Multi Strings can be solved in time $((|\Sigma| + 1)^{\mathcal{O}(k^2)}(\Delta k)^{\mathcal{O}(k^2)} \log(L) + \mathcal{O}((|\Sigma| + 1)^k kL)$. Subsequent algorithms for n-fold ILP applied on the program they consider, including [19, 6], result in algorithms that run in time $(k\Delta)^{\mathcal{O}(k^2)}(|\Sigma| + 1)^{\mathcal{O}(k)} + \mathcal{O}((|\Sigma| + 1)^k kL)$. Next, we discuss the application of existing algorithms in the above program. Using the algorithms for standard ILPs, including [22, 21], results in algorithms that are double exponential in k. Using the algorithm by Cslovjecsek et al. [6] results in an algorithm that runs in time $(k\Delta)^{\mathcal{O}(k^2)}(|\Sigma| + 1)^{\mathcal{O}(k)} + \mathcal{O}((|\Sigma| + 1)^k kL)$.

The problem Closest String is the special case of δ-Multi Strings where $\beta = 0$, the distance function δ is the Hamming distance, and $d_i = 0$ and $D_i = d$ for each $i \in [k]$. Applying Theorem 9, we obtain an algorithm for Closest String that runs in time $(k)^{\mathcal{O}(k^2)} \cdot L^2$. In terms of the dependence on k, this matches the current best algorithms [11, 25].

When $\Sigma = \{0, 1\}$ in the Closest String problem, Theorem 9 gives an algorithm that runs in time $2^{\mathcal{O}(k^2)} \cdot L^2$. We note that this is an improvement over applying the existing algorithms for n-fold ILP, including [19, 6] which have a $k^{\mathcal{O}(k^2)}$ term in the running time. To the best of our knowledge, we are the first to exhibit an algorithm with $2^{\mathcal{O}(k^2)}$ dependence on k. We note that this matches the lower bound hypothesized by Rohwedder and Wegrzycki [31] for the problem. Thus, we have the following.

▶ **Corollary 10.** *Closest String can be solved in time $(k)^{\mathcal{O}(k^2)} \cdot L^{\mathcal{O}(1)}$. Moreover, Binary Closest String can be solved in time $2^{\mathcal{O}(k^2)} \cdot L^{\mathcal{O}(1)}$.*

5.3 Equitable coloring parameterized by vertex cover

We consider the Equitable Coloring problem parameterized by the vertex cover number k. We note that Gomes et al. [14] show that the problem can be solved in time $2^{\mathcal{O}(k \log k)} \cdot |V(G)|^{\mathcal{O}(1)}$ using a flow-based approach. Though we don't match their running time, we still present an ILP to exhibit the applicability of our algorithm.

Given a graph $G = (V, E)$, a *vertex coloring* of G is a function $c : V(G) \mapsto \mathbb{N}$ if for any $(u, v) \in E(G)$ we have $c(u) \neq c(v)$. A vertex v is said to be *colored* i if $c(v) = i$. For each i, the set of vertices colored i form the i^{th} color class V_i. We study the following problem.

Equitable Coloring
Input: A graph $G = (V, E)$ and a positive integer h
Question: Is there a vertex coloring c of G using at most h colors such that the sizes of any two color classes differ by at most 1.

A *vertex cover* in G is a subset of vertices whose deletion results in an edgeless graph (also called an *independent set*). The *vertex cover number* of G is the size of a smallest vertex cover in G. Fiala et al. [13] show that Equitable Coloring is FPT when with respect to the vertex cover number k. They study the two cases of $h \leq k$ and $h \geq k$ separately. For the sake of presentation, we only cover the first case, and we note that the second case is dealt with in an almost identical manner and the claimed running time holds.

We assume that we are given a vertex cover W of size k: note that such a W can be computed in time $2^{\mathcal{O}(k)}$ [8]. Let $\{I_1, \ldots, I_s\}$ denote the partition of the independent set $V(G) \setminus W$ according to their neighborhoods. Observe that $s \leq 2^k$.

Let $w = \lfloor \frac{|V(G)|}{h} \rfloor$, $a = |V(G)| - hw$, and $b = h - a$. For each proper coloring $(V_1, \ldots, V_h)$ of W, they construct a system of linear inequalities with sh variables $x_j^{(i)}$, $i \in [s]$ and $j \in [h]$, where $x_j^{(i)}$ indicates the number of vertices of color j in the set I_i:

1. $x_j^{(i)} \geq 0$;
2. $x_j^{(i)} = 0$, if color j is used in $N(I_i)$. Note that the non-negativity of the variables implies that this can equivalently be expressed as a single constraint $\sum_{i,j} x_j^{(i)} = 0$ where the summation is over $i \in [s], j \in [h]$ such that color j is used in $N(I_i)$;
3. $x_j^{(1)} + \cdots + x_j^{(s)} = t + 1 - |W \cap V_j|$ if $j \in \{1 \ldots, a\}$;
4. $x_j^{(1)} + \cdots + x_j^{(s)} = t - |W \cap V_j|$ if $j \in \{a, \ldots, h\}$;
5. $x_1^{(i)} + \cdots + x_h^{(i)} = |I_i|$ for each $i \in [s]$.

We claim that the above integer program is of the form $(\mathbf{P_2})$. The sh variables are partitioned into s blocks of size h. The constraints written in Items 2–4 form the $h+1$ global constraints. The constraints written in Item 5 form the local constraints. The parameters corresponding to the above program are $r = k+1$, $t = h \leq |V(G)|$, $n = s \leq 2^k$, $\Delta = 1$, and $q = \sum_{i \in [s]} |I_i| = |V(G)| - k$. Recall that for each k-coloring of W an integer program of the above kind is created. This can be done in time $2^{\mathcal{O}(k)} \cdot |V(G)|^{\mathcal{O}(1)}$. Thus, the overall running time is given by summing over all k^k colorings the time to create and solve the corresponding program. Applying Corollary 7, we have the following.

▶ **Theorem 11.** *Equitable Coloring can be solved in time* $2^{\mathcal{O}(k^2)} \cdot |V(G)|^{\mathcal{O}(1)}$.

We note that using existing algorithms for n-fold ILP, including [19, 6], results in a $k^{\mathcal{O}(k^2)}$ dependence on k while we only have a $2^{\mathcal{O}(k^2)}$ dependence.

References

1 Gautam Appa, Balázs Kotnyek, Konstantinos Papalamprou, and Leonidas S. Pitsoulis. Optimization with binet matrices. *Oper. Res. Lett.*, 35(3):345–352, 2007. doi:10.1016/J.ORL.2006.04.003.

2 Stephan Artmann, Robert Weismantel, and Rico Zenklusen. A strongly polynomial algorithm for bimodular integer linear programming. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017*, pages 1206–1219. ACM, 2017. doi:10.1145/3055399.3055473.

3 Imre Bárány. *On the Power of Linear Dependencies*, pages 31–45. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. doi:10.1007/978-3-540-85221-6_1.

4 Robert Bredereck, Jiehua Chen, Sepp Hartung, Stefan Kratsch, Rolf Niedermeier, Ondrej Suchý, and Gerhard J. Woeginger. A multivariate complexity analysis of lobbying in multiple referenda. *J. Artif. Intell. Res.*, 50:409–446, 2014. doi:10.1613/JAIR.4285.

5 Robin Christian, Mike Fellows, Frances Rosamond, and Arkadii Slinko. On complexity of lobbying in multiple referenda. *Review of Economic Design*, 11(3):217–224, November 2007. doi:10.1007/s10058-007-0028-1.

6 Jana Cslovjecsek, Friedrich Eisenbrand, Christoph Hunkenschröder, Lars Rohwedder, and Robert Weismantel. Block-structured integer and linear programming in strongly polynomial and near linear time. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 1666–1681. SIAM, 2021. doi:10.1137/1.9781611976465.101.

7 Jana Cslovjecsek, Martin Koutecký, Alexandra Lassota, Michal Pilipczuk, and Adam Polak. Parameterized algorithms for block-structured integer programs with large entries. In David P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024*, pages 740–751. SIAM, 2024. doi:10.1137/1.9781611977912.29.

8 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.

9 Friedrich Eisenbrand, Christoph Hunkenschröder, and Kim-Manuel Klein. Faster algorithms for integer programs with block structure. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018*, volume 107 of *LIPIcs*, pages 49:1–49:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.ICALP.2018.49.

10 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.

11 Friedrich Eisenbrand, Lars Rohwedder, and Karol Wegrzycki. Sensitivity, proximity and FPT algorithms for exact matroid problems. In *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024*, pages 1610–1620. IEEE, 2024. doi:10.1109/FOCS61266.2024.00100.

12 Friedrich Eisenbrand and Robert Weismantel. Proximity results and faster algorithms for integer programming using the Steinitz Lemma. *ACM Trans. Algorithms*, 16(1):5:1–5:14, 2020. doi:10.1145/3340322.

13 Jirí Fiala, Petr A. Golovach, and Jan Kratochvíl. Parameterized complexity of coloring problems: Treewidth versus vertex cover. *Theor. Comput. Sci.*, 412(23):2513–2523, 2011. doi:10.1016/J.TCS.2010.10.043.

14 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.

15 V. S. Grinberg and S. V. Sevast'yanov. Value of the steinitz constant. *Functional Analysis and Its Applications*, 14(2):125–126, April 1980. doi:10.1007/BF01086559.

16 Raymond Hemmecke, Shmuel Onn, and Lyubov Romanchuk. n-fold integer programming in cubic time. *Math. Program.*, 137(1-2):325–341, 2013. doi:10.1007/S10107-011-0490-Y.

17 Klaus Jansen, Kai Kahler, Lis Pirotton, and Malte Tutas. Improving the parameter dependency for high-multiplicity scheduling on uniform machines. *CoRR*, abs/2409.04212, 2024. doi:10.48550/arXiv.2409.04212.

18 Klaus Jansen, Kai Kahler, Lis Pirotton, and Malte Tutas. New algorithm for combinatorial n-folds and applications, 2025. arXiv:2409.04212.

19 Klaus Jansen, Alexandra Lassota, and Lars Rohwedder. Near-linear time algorithm for n-fold ILPs via color coding. *SIAM J. Discret. Math.*, 34(4):2282–2299, 2020. doi:10.1137/19M1303873.

20 Klaus Jansen and Lars Rohwedder. On integer programming and convolution. In Avrim Blum, editor, *10th Innovations in Theoretical Computer Science Conference, ITCS 2019*, volume 124 of *LIPIcs*, pages 43:1–43:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.ITCS.2019.43.

21 Klaus Jansen and Lars Rohwedder. On integer programming, discrepancy, and convolution. *Math. Oper. Res.*, 48(3):1481–1495, 2023. doi:10.1287/MOOR.2022.1308.

22 Ravi Kannan. Minkowski's convex body theorem and integer programming. *Math. Oper. Res.*, 12(3):415–440, 1987. doi:10.1287/MOOR.12.3.415.

23 Dusan Knop and Martin Koutecký. Scheduling meets n-fold integer programming. *J. Sched.*, 21(5):493–503, 2018. doi:10.1007/S10951-017-0550-0.

24 Dusan Knop, Martin Koutecký, Asaf Levin, Matthias Mnich, and Shmuel Onn. High-multiplicity n-fold IP via configuration LP. *Math. Program.*, 200(1):199–227, 2023. doi:10.1007/S10107-022-01882-9.

25 Dusan Knop, Martin Koutecký, and Matthias Mnich. Combinatorial n-fold integer programming and applications. *Math. Program.*, 184(1):1–34, 2020. doi:10.1007/S10107-019-01402-2.

26 Dusan Knop, Martin Koutecký, and Matthias Mnich. Voting and bribing in single-exponential time. *ACM Trans. Economics and Comput.*, 8(3):12:1–12:28, 2020. doi:10.1145/3396855.

27 Martin Koutecký, Asaf Levin, and Shmuel Onn. A parameterized strongly polynomial algorithm for block structured integer programs. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018*, volume 107 of *LIPIcs*, pages 85:1–85:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.ICALP.2018.85.

28 Trung Thanh Nguyen and Jörg Rothe. Complexity results and exact algorithms for fair division of indivisible items: a survey. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, IJCAI '23, 2023. doi:10.24963/ijcai.2023/754.

29 Christos H. Papadimitriou. On the complexity of integer programming. *J. ACM*, 28(4):765–768, 1981. doi:10.1145/322276.322287.

30 Lars Rohwedder. Eth-tight FPT algorithm for makespan minimization on uniform machines. In Keren Censor-Hillel, Fabrizio Grandoni, Joël Ouaknine, and Gabriele Puppis, editors, *52nd International Colloquium on Automata, Languages, and Programming, ICALP 2025, July 8-11, 2025*, volume 334 of *LIPIcs*, pages 126:1–126:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPICS.ICALP.2025.126.

31 Lars Rohwedder and Karol Wegrzycki. Fine-grained equivalence for problems related to integer linear programming. In Raghu Meka, editor, *16th Innovations in Theoretical Computer Science Conference, ITCS 2025*, volume 325 of *LIPIcs*, pages 83:1–83:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPICS.ITCS.2025.83.

32 Sergey Vasil'evich Sevast'janov. on some geometric methods in scheduling theory: A survey. *Discret. Appl. Math.*, 55(1):59–82, 1994. doi:10.1016/0166-218X(94)90036-1.

33 Ernst Steinitz. Bedingt konvergente reihen und konvexe systeme. *Journal für die reine und angewandte Mathematik*, 1913(143):128–176, 1913. doi:10.1515/crll.1913.143.128.

New Algorithm for Combinatorial n-Folds and Applications

Klaus Jansen
Department of Computer Science, Kiel University, Germany

Kai Kahler
Department of Computer Science, Kiel University, Germany

Lis Pirotton
Department of Computer Science, Kiel University, Germany

Malte Tutas
Department of Computer Science, Kiel University, Germany

Abstract

Block-structured integer linear programs (ILPs) play an important role in various application fields. We address n-fold ILPs where the matrix $\mathcal{A}$ has a specific structure, i.e., where the blocks in the lower part of $\mathcal{A}$ consist only of the row vectors $(1, \ldots, 1)$. In this paper, we propose an approach tailored to exactly these combinatorial n-folds. We utilize a divide and conquer approach to separate the original problem such that the right-hand side iteratively decreases in size. We show that this decrease in size can be calculated such that we only need to consider a bounded amount of possible right-hand sides. This, in turn, lets us efficiently combine solutions of the smaller right-hand sides to solve the original problem. We can decide the feasibility of, and also optimally solve, such problems in time $(nr\Delta)^{O(r)} \log(\|b\|_\infty)$, where n is the number of blocks, r the number of rows in the upper blocks and $\Delta = \|A\|_\infty$.

We complement the algorithm by discussing applications of the n-fold ILPs with the specific structure we require. We consider the problems of (i) scheduling on uniform machines, (ii) closest string and (iii) (graph) imbalance. Regarding (i), our algorithm results in running times of $p_{\max}^{O(d)}|I|^{O(1)}$, matching a lower bound derived via ETH. For (ii) we achieve running times matching the current state-of-the-art in the general case. In contrast to the state-of-the-art, our result can leverage a bounded number of column-types to yield an improved running time. For (iii), we improve the parameter dependency on the size of the vertex cover.

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

Keywords and phrases integer linear programming, n-fold, parameterized complexity, scheduling, uniform machines

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.15

Related Version *Full Version*: `https://arxiv.org/abs/2409.04212` [17]

Funding Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project numbers 453769249 and 528381760.

Acknowledgements We thank Martin Koutecký for the helpful discussions regarding the topic of $P\|C_{\max}$.

1 Introduction

At an abstract level, computer science is about solving hard problems. In a seminal work, Karp formulated a list of twenty-one NP-hard problems [24]. Among these is the problem of integer linear programming. An integer linear program (ILP) in standard form is defined by

$$\max\{c^T x \mid \mathcal{A}x = b, x \in \mathbb{Z}_{\geq 0}\},$$

© Klaus Jansen, Kai Kahler, Lis Pirotton, and Malte Tutas;
licensed under Creative Commons License CC-BY 4.0

with constraint matrix $\mathcal{A} \in \mathbb{Z}^{m \times n}$, right-hand side (RHS) $b \in \mathbb{Z}^m$ and objective function vector $c \in \mathbb{Z}^n$. The constraints $Ax = b$ and the objective function $c^T x$ are required to be linear functions. Such integer programs have been the subject of vast amounts of research, such as [35, 22, 36, 21], with the latter two results representing the current state of the art of running times regarding n and m respectively. However, with integer programming being an NP-hard problem, research has flourished into different structures of integer programs, which utilize a more specific structure to yield improved running times. One such structure is the n-fold ILP.

n-fold ILP

Let $r, s, t, n \in \mathbb{Z}_{\geq 0}$. An n-fold ILP is an integer linear program with RHS vector $b \in \mathbb{Z}^{r+ns}$ and with a constraint matrix of the form

$$\begin{pmatrix} A_1 & A_2 & \dots & A_n \\ B_1 & 0 & \dots & 0 \\ 0 & B_2 & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \dots & 0 & B_n \end{pmatrix},$$

where $A_1, \dots, A_n \in \mathbb{Z}^{r \times t}$ and $B_1, \dots, B_n \in \mathbb{Z}^{s \times t}$ are matrices, and $x = (x_1, \dots, x_n)$ is the desired solution. In other words, if we delete the first r constraints, which are called the *global* constraints, the problem decomposes into independent programs, which are called *local*. If Δ is the largest absolute value in A_i and B_i, n-fold integer programs can be solved in time $(nt)^{(1+o(1))} \cdot 2^{O(rs^2)}(rs\Delta)^{O(r^2s+s^2)}$ [8]. This algorithm comes as the latest development in a series of improvements spanning almost twenty years [1, 8, 9, 32, 10, 16, 20, 30]. n-fold integer programs have already found applications in countless problems, often yielding improved running times over other techniques. As the total number of applications is too large, we refer the reader to [7, 32, 14, 19, 25, 26, 28] for some examples.

In the following, we focus on combinatorial n-fold ILPs where we allow blocks of different width, i.e., $t \in \mathbb{Z}^n_{\geq 0}$ is a vector and therefore, the block $i \in [n]$ has t_i columns. Also, we have $B_i = (1, \dots, 1)^T = \mathbb{1}_{t_i}$ for all $i \in [n]$ which implies $s = 1$. Thus, we have the following ILP

$$\mathcal{A}x = \begin{pmatrix} A_1 & A_2 & \dots & A_n \\ \mathbb{1}_{t_1} & 0 & \dots & 0 \\ 0 & \mathbb{1}_{t_2} & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \dots & 0 & \mathbb{1}_{t_n} \end{pmatrix} x = b, \qquad x \in \mathbb{Z}^h_{\geq 0} \tag{♠}$$

with $A_i \in \mathbb{Z}^{r \times t_i}$ for all $i \in [n]$, $b \in \mathbb{Z}^{r+n}$ and where $h := \sum_{i \in [n]} t_i$ is the total number of columns in $\mathcal{A}$. The currently best known algorithms for such combinatorial n-fold ILPs have parameter dependency $(r\Delta)^{O(r^2)}$ [8, 28]. In our work, we often consider the upper part of the vector b differently from its lower part. Therefore, we introduce the following notation: $b = (b^\uparrow, b^\downarrow)^T$ with $b^\uparrow \in \mathbb{Z}^r$ and $b^\downarrow \in \mathbb{Z}^n$. Also, define $b^\uparrow_{\max} := \max_{j \in [r]} b^\uparrow_j$, $b^\downarrow_{\max} := \max_{k \in [n]} b^\downarrow_k$ and $b_{\text{def}} := \min(b^\uparrow_{\max}, b^\downarrow_{\max})$.

Our contribution

In this paper, we provide a novel approach to solving such combinatorial n-fold ILPs. While the idea of the algorithm we provide is inspired by Jansen and Rohwedder [21], we note that we utilize the structure of n-fold ILPs to generate different techniques to split solutions apart into smaller sub-solutions. Using these techniques, we achieve the following result.

▶ **Theorem 1.** *Combinatorial n-fold ILPs (♠) with respect to the objective of maximizing $c^T x, c \in \mathbb{Z}^h_{\geq 0}$ and where Δ is the largest absolute value in $\mathcal{A}$ can be solved in time $(nr\Delta)^{O(r)} \log(b_{def})$.*

Comparatively, the algorithm in [21] yields a running time of $O(\sqrt{r+n}\Delta)^{2(r+n)}+O(h\cdot(r+n))$ when applied to our setting, while the state-of-the-art algorithm [8] achieves a running time of $(n\|t\|_\infty)^{(1+o(1))} \cdot 2^{O(r)}(r\Delta)^{O(r^2)}$.

To complement the general techniques we provide to solve combinatorial n-fold ILPs, we show how our result can be applied to common problems. First, we consider scheduling on uniform machines with the objective of makespan minimization ($Q\|C_{\max}$) and Santa Claus ($Q\|C_{\min}$). After publication of a preliminary version of our results, Rohwedder [37] recently bounded Δ by $p_{\max}^{O(1)}$ for $Q\|C_{\max}$, where $p_{\max}$ is the largest processing time. They also provide an (almost) tight algorithm for $Q\|C_{\max}$ with running time $(dp_{\max})^{O(d)}(h+M)^{O(1)}|I|^{O(1)}$, where d is the number of job-types, h is the total number of columns in the configuration ILP and M is the total number of machines. Replacing a subroutine in [37] by our algorithm improves their result to $(dp_{\max})^{O(d)} \log(m_{\text{def}})|I|^{O(1)}$, where $m_{\text{def}} := \min(n_{\max}, m_{\max})$ i.e., we achieve a better result with respect to parameter h and M. This answers the open question by Koutecký and Zink in [31]. They pose the question whether $Q\|C_{\max}$ can be solved in time $p_{\max}^{O(d)}|I|^{O(1)}$. In more detail, the calculation of the Multi-Choice IP in [37] has a running time of $(nr\Delta)^{O(r)}(h+\|b^{\downarrow}\|_1)$, while our n-fold algorithm runs in $(nr\Delta)^{O(r)} \log(b_{\text{def}})$, i.e., solves these Multi-Choice IPs more quickly.

We also manage to extend these techniques to $Q\|C_{\min}$. This yields running times that are (almost) tight by an ETH-lower bound.

▶ **Theorem 2.** *$Q\|C_{\max}$ and $Q\|C_{\min}$ can be solved in time $p_{\max}^{O(d)}|I|^{O(1)}$.*

We also apply our algorithm to the Closest String problem and the Imbalance problem (see Section 4 for formal definitions). The current best running times [28] for both problems have a quadratic dependency of k in the exponent, where k is the number of considered strings (Closest String), respectively the size of the vertex cover (Imbalance). When the number of column-types (Closest String), respectively vertex-types (Imbalance) is bounded by T, our algorithm achieves the following results.

▶ **Corollary 3.** *The Closest String problem can be solved in time $((T+1)k)^{O(k)} \log(L)$.*

▶ **Corollary 4.** *The Imbalance problem with n vertices can be solved in time $((T+2)k)^{O(k)} \log(kn)$, where k is the size of a vertex cover and T is the number of vertex-types.*

Generally, T is bounded by k^k for Closest String. In this case, we meet the state-of-the-art parameter dependency. For Imbalance, T may take any integer value up to 2^k. The resulting parameter dependency of 2^{k^2} improves on the state-of-the-art k^{k^2} which is the application of [28] to the problem. Additionally, when T is bounded by $k^{O(1)}$, we are able to reduce the dependency in the exponent to linear for both problems. When applying Rohwedder's algorithm for the Multi-Choice IP [37, Theorem 2] to those problems, the resulting running times are $((T+1)k)^{O(k)}O(T+dk)$, where d is the upper bound on string distance (Closest String) and $((T+2)k)^{O(k)}O(Tk+nk)$ (Imbalance).

We expect that our techniques can be extended to several other problems that can be formulated as a combinatorial n-fold ILP as well, such as the string problems given in [28].

Related Work

Integer linear programming is one of the twenty-one problems originally proven to be NP-hard by Karp in his seminal work [24]. Since then, ILPs have been used to solve countless NP-hard problems, for examples see e.g. [14]. The currently best known running times can

be separated by their parameterization. When considering a small number of constraints m, the best known running time is given by Jansen and Rohwedder [21], and runs in time $O(\sqrt{m}\Delta)^{(1+o(1))m} + O(nm)$. For a small number of variables n, the current best known algorithm is given by Reis and Rothvoss [36], with a running time of $(\log n)^{O(n)} \cdot |\mathcal{A}|^{O(1)}$.

With the general ILP being a hard problem, research into certain substructures that are more easily solvable began. For an overview of this process, see e.g., [11, 14]. Of particular interest to this paper is the n-fold ILP, whose current best known algorithm is given by Cslovjecsek, Eisenbrand, Hunkenschröder, Rohwedder and Weismantel [8]. Their algorithm runs in time $(nt)^{(1+o(1))} \cdot 2^{O(rs^2)}(rs\Delta)^{O(r^2s+s^2)}$ and is the latest result in a series of improvements that spans close to twenty years, see [9, 32, 10, 16, 20, 30] for works complementing this line of research.

Scheduling is one of the most well-known operations research problems. We now specifically turn towards uniform machines. By formulating the problem as an n-fold ILP, Knop and Koutecký [29] showed that $Q\|C_{\max}$ can be solved in time $p_{\max}^{O(p_{\max}^2)}|I|^{O(1)}$ if the number of machines is encoded in unary. By solving a relaxation to schedule many of the jobs in advance and then applying a dynamic program, Koutecký and Zink [31] improved this to $p_{\max}^{O(d^2)}|I|^{O(1)}$, still with the number of machines M encoded in unary. Leveraging the huge-n-fold machinery, Knop et al. [27] get rid of the assumption on M and obtain a $p_{\max}^{O(d^2)}|I|^{O(1)}$-time algorithm. Koutecký and Zink [31] pose the open question whether the dependency $p_{\max}^{O(d^2)}$ for the objective $Q\|C_{\max}$ can be improved, as it has been for identical machines. In a recent result, Rohwedder [37] gives an algorithm for $Q\|C_{\max}$ running in time $p_{\max}^{O(d)}|I|^{O(1)}$. We note that our work has been compiled independently and prior to Rohwedders result, but we can leverage his novel ideas to reduce the value of Δ to guarantee a similar result. Chen, Jansen and Zhang show that, for a given makespan T, there is no algorithm solving $P\|C_{\max}$ in time $2^{T^{1-\delta}}|I|^{O(1)}$ [6]. Jansen, Kahler and Zwanger extended these results to derive a lower bound of $p_{\max}^{O(d^{1-\delta})}|I|^{O(1)}$ [18]. As $Q\|C_{\max}$ is a generalization of $P\|C_{\max}$, these results extend to it.

Regarding the Closest String problem with k input strings of length L, Gramm et al. [15] formulated the problem as an ILP of dimension $k^{O(k)}$. This led to an algorithm with running time $2^{2^{O(k\log(k))}}O(\log(L))$. Knop et al. [28] improved this double-exponential dependency to the currently best known algorithm with running time $k^{O(k^2)}O(\log(L))$. Rohwedder and Węgrzycki [38] showed that the Closest String problem (with binary alphabet) cannot be solved in time $2^{O(k^{2-\varepsilon})}\text{poly}(L)$ with $\varepsilon > 0$ unless the ILP Hypothesis fails. The ILP Hypothesis states that for every $\varepsilon > 0$, there is no $2^{O(m^{2-\varepsilon})}\text{poly}(L)$-time algorithm for ILPs with $\Delta = O(1)$.

Imbalance is known to be NP-complete for various special graph classes [4, 23]. Fellows, Lokshtanov, Misra, Rosamond and Saurabh [13] showed that the problem is fixed-parameter tractable (FPT) parameterized by the size k of the vertex cover. Their algorithm has a double exponential running time. There also exist FPT algorithms for this problem parameterized by other parameters, for instance imbalance [33], neighborhood diversity [2] and the combined parameter of treewidth and maximum degree [33]. Misra and Mittal [34] showed that Imbalance is in XP when parameterized by twin cover, and FPT when parameterized by the twin cover and the size of the largest clique outside the twin cover.

2 Overview

In this section, we give an overview of the paper and the techniques within. We begin by giving some concepts essential to the functionality of our algorithm. Then, we prove the feasibility version of Theorem 1 under the assumption that $\mathcal{A}$ contains only non-negative

entries. At its core, we use a divide and conquer approach to separate the entire problem into smaller sub-problems, which we can then solve more quickly via dynamic programming. Then, we combine these small solutions such that they solve iteratively larger sub-problems, again requiring a novel structural result to show that such combinations are feasible. A core strength of this dynamic programming formulation and approach is that we only need to solve the dynamic program once for each recombination step. This procedure can be iterated until we have solved the entire n-fold ILP. After that, we show how we can reduce the general problem which also allows negative coefficients to the special case where $\mathcal{A}$ is non-negative. After that, we prove Theorem 1.

The algorithm we provide is complemented by a discussion on its applications to relevant problems. We begin this by elaborating on the problem of scheduling on uniform machines in Section 4. Here, we can adapt some novel concepts to our techniques such that we achieve a running time that is (almost) tight according to a lower bound derived via the exponential time hypothesis. This yields Theorem 2. Finally, in the full version [17] we provide a brief outlook onto other relevant problems, the Closest String problem which results in Corollary 3 and the Imbalance problem which results in Corollary 4.

The full proofs as well as omitted statements can be found in the full version [17].

Preliminaries

Before we define concepts essential to the algorithm and its functionality, we first specify some notation. We denote $[k] := \{i \in \mathbb{Z}_{\geq 0} \mid 1 \leq i \leq k\}$ and use base 2 logarithms, i.e., we assume $\log(2) = 1$. A vector that contains a certain value in all entries is stylized, e.g. the zero vector 0_d is written as $\mathbb{0}_d$. We omit the dimension d if it is apparent from the context. For vector v, we refer to its maximum value $\|v\|_\infty$ with $v_{\max}$. The support of a d-dimensional vector v is the set of its indices with a non-zero entry. It is denoted by $\operatorname{supp}(v) = \{i \in [d] \mid v_i \neq 0\}$. We define a *brick* $x^{(i)} \in \mathbb{Z}^{t_i}$ as the i-th block of the solution vector of (♠), i.e., we have $x = \left(x^{(1)}x^{(2)}\dots x^{(n)}\right)^T$, with $x^{(i)} \in \mathbb{Z}^{t_i}$ for all $i \in [n]$.

Support of a Solution

By applying the result by Eisenbrand and Shmonin [12, Theorem 1 (ii)] to the n-fold ILP, one can show the following support bound:

▶ **Lemma 5.** *Consider the ILP (♠) and let Δ be the largest absolute value of any entry in the coefficient matrix $\mathcal{A}$. Then there exists a solution $x = (x_1, \dots, x_n)^T$ to (♠) with*

$$|\operatorname{supp}(x_k)| \leq 2(r+1)\log(4(r+1)\Delta), \qquad \forall k \in [n].$$

Proof. Let $y = (y_1, \dots, y_n)^T$ be a solution to (♠). Then for each $k \in [n]$, the ILP

$$\begin{pmatrix} A_k \\ \mathbb{1}^T \end{pmatrix} x_k = \begin{pmatrix} A_k y_k \\ b_{r+k} \end{pmatrix} \quad x_k \in \mathbb{Z}^c_{\geq 0}$$

is feasible and we can apply the support bound by Eisenbrand and Shmonin [12, Theorem 1 (ii)]. Hence, there exists a solution x_k with $|\operatorname{supp}(x_k)| \leq 2(r+1)\log(4(r+1)\Delta)$. Combining these solutions for all $k \in [n]$ yields a solution x with the desired property. ◀

Note that this bound only holds for the feasibility problem. Using the support bound by Eisenbrand and Shmonin [12, Corollary 5 (ii)], the same technique achieves the following result $|\operatorname{supp}(x_k)| \leq 2(r+2)(\log(r+2)+\Delta+2)$. Other bounds, for instance [3], can also be applied in our approach and may achieve better results in specific application settings.

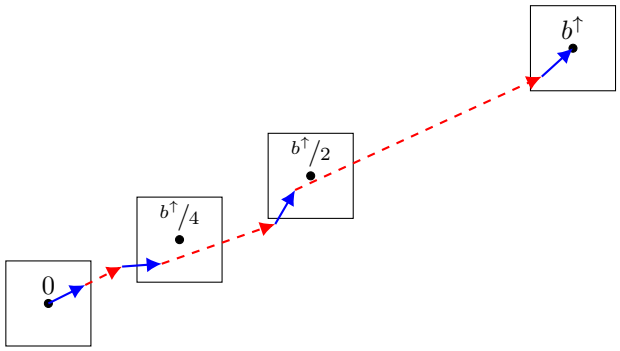


Figure 1 Example of the procedure with four iterations. The continuous (blue) arrows indicate a feasible upper RHS of the sub-problem in the current iteration. The dashed (red) arrows double the intermediate solution of the previous iteration.

3 Combinatorial n-fold Algorithm

On a high level, our algorithm implements an elegant idea. Instead of solving the – often times large – entire n-fold ILP in one fell swoop, we manage to reduce it to a set of smaller problems, which we then use to solve the large problem. More specifically, we show that there exists a feasible solution to (♠) that can be constructed by doubling a solution *near* $b^\uparrow/2$ and adding the solution of a small sub-problem. This concept can be repeated iteratively, i.e., a solution near $b^\uparrow/2$ can be computed by doubling a solution near $b^\uparrow/4$ and adding the solution of another small sub-problem, and so on. We repeat this until the remaining problem only contains a small upper RHS $b^\uparrow$. For an illustration of this process, see Figure 1. A core strength of this approach is that we only need to compute this dynamic program once for every iteration, independent of the amount of candidate solutions.

Taking a closer look, we use insights generated from the structure of our repeatedly doubled intermediary solutions to uniquely determine the lower RHS $b^\downarrow$ at every step of the algorithm. As we double solutions in each step, we can calculate the total number of iterations our algorithm needs. We denote this number by $\mathcal{I} \in \mathbb{Z}_{\geq 1}$. Later in this section, we give the exact value of $\mathcal{I}$ and provide a proof that our algorithm needs exactly $\mathcal{I}$ iterations.

For any iteration $i \in [\mathcal{I}]$, we denote the lower RHS by $b^{\downarrow(i)}$. In the first iteration, and every time we solve a small sub-problem, we know by Lemma 5 that the support of the individual bricks is bounded by

$$K := \lfloor 2 \cdot (r+1) \cdot \log(4 \cdot (r+1) \cdot \Delta) \rfloor, \tag{1}$$

which also limits the upper RHS, which we refer to as D, of these sub-problems. Later, we show how this bound D is determined. The more detailed process is illustrated in Figure 2. Here, the continuous (blue) arrow contains the solution to the small sub-problem, while the dashed (red) arrow represents the doubling of an earlier solution.

We begin the description of our algorithm by showing that the lower RHS in each of these iterations is uniquely determinable. We further show that we can indeed find solutions close to each $b^\uparrow/2^{\mathcal{I}-i}$ that can be doubled to produce a feasible solution. Finally, we show how to utilize these insights to construct a feasible schedule while only needing to iteratively solve small sub-problems.

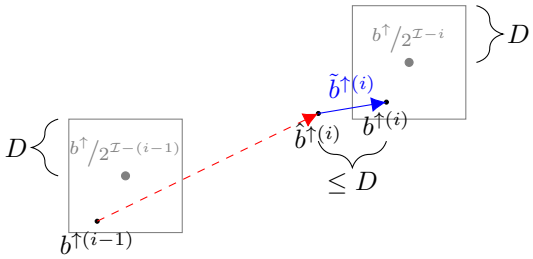


Figure 2 Zoom in on a single iteration i including an illustration of the designation of the upper RHS $b^{\uparrow(i-1)}, \hat{b}^{\uparrow(i)}, \tilde{b}^{\uparrow(i)}$ and $b^{\uparrow(i)}$ and the upper bound D of the box sizes and $\|\tilde{b}^{\uparrow(i)}\|_\infty$.

3.1 Structure of Intermediary Solutions

In this section, we show that if (♠) is feasible, there exists a feasible upper RHS near $b^{\uparrow}/2^{\mathcal{I}-i}$, for all $i \in [\mathcal{I}]$, which then can be extended to solve main n-fold. We say that a RHS b or upper RHS $b^{\uparrow}$ is feasible if and only if there exists a solution x to $\mathcal{A}x = (b^{\uparrow}, b^{\downarrow})^T$. We bound the size of the RHS in the small sub-problems needed to be solved in order to extend these intermediary solutions in each step.

Within this section, we view the problem top-down, i.e., in each iteration $i \in [\mathcal{I}]$, we divide the problem into two sub-problems. The first sub-problem, which we call *parity constrained*, contains only even entries in its lower RHS $\hat{b}^{\downarrow}$. Because of this, that sub-problem can then be divided in two equal parts which we solve inductively. We call the second sub-problem, with lower RHS $\tilde{b}^{\downarrow}$, *small*. The *small* sub-problem fulfills $\|\tilde{b}^{\downarrow}\|_\infty \leq K$ and can therefore be solved efficiently using dynamic programming.

For a given iteration i, we denote the RHS of the considered problem by $b^{(i)}$. Note that we have $b^{(\mathcal{I})} = b$. The division into the two sub-problems implies $b^{(i)} = \hat{b}^{(i)} + \tilde{b}^{(i)}$. As we divide parity constrained problem into two equal problems and then repeat the procedure, we have $b^{(i-1)} = \hat{b}^{(i)}/2$. We ensure, that the upper RHS $b^{\uparrow(i)}$ also contains only even entries.

With this notation in mind, we begin by determining the lower RHS values for each iteration and their sub-problems. To determine the value of the k-th entry in the lower RHS of the small sub-problems $\tilde{b}^{\downarrow(i)}$, we consider the corresponding entry $b_k^{\downarrow(i)}$. If that entry is small, that is if $b_k^{\downarrow(i)} \leq K$, we set $\tilde{b}_k^{\downarrow(i)} = b_k^{\downarrow(i)}$. This implies $\hat{b}^{\downarrow(i)} = b_k^{\downarrow(i)} - \tilde{b}_k^{\downarrow(i)} = 0$ and $b_k^{\downarrow(i-1)} = \hat{b}_k^{\downarrow(i)}/2 = 0$. Therefore, for all following iterations this entry remains 0, i.e, $b_k^{\downarrow(i')} = \tilde{b}_k^{\downarrow(i')} = 0$ and $\hat{b}_k^{\downarrow(i')} = 0$ for all $i' < i$. However, if $b_k^{\downarrow(i)}$ is large, that is if $b_k^{\downarrow(i)} > K$, we set $\tilde{b}_k^{\downarrow(i)}$ to K or $K-1$ such that $\hat{b}_k^{\downarrow(i)} = b_k^{\downarrow(i)} - \tilde{b}_k^{\downarrow(i)}$ is even. Thus, the parity constrained entry is larger than 0 and can be divided by 2. Now, set $b_k^{\downarrow(i-1)} = \hat{b}_k^{\downarrow(i)}/2$ and repeat the procedure for the next smaller iteration.

More formally, for known value of $b_k^{\downarrow(i+1)} > K$ we can derive $b_k^{\downarrow(i)}$ as follows:

$$b_k^{\downarrow(i)} = \frac{b_k^{\downarrow(i+1)} - (K + z_k^{(i+1)})}{2} \quad \text{with} \tag{2}$$

$$z_k^{(i+1)} = \begin{cases} 0, & \text{if } \left(b_k^{\downarrow(i+1)} \mod 2\right) = (K \mod 2) \\ 1, & \text{otherwise.} \end{cases}$$

The following lemmas derive that, despite of the uncertainty due to the parity-constraint, the lower RHS in each iteration can be uniquely determined from the input $(\mathcal{A}, b)$ to (♠).

▶ **Lemma 6** (✂). *Let $b_k^{\downarrow} \in \mathbb{Z}^n_{\geq 0}$ and $k \in [n]$. Then $b_k^{\downarrow(i)}$ is integer for all $i \in [\mathcal{I}]$, where values are determined by the procedure iteratively applying* (2).

Using this property and (2), we achieve the desired result by elegantly formulating the term as a continued fraction and using a property of the geometric sum.

▶ **Lemma 7** (✂). *Let $b_k^{\downarrow} \in \mathbb{Z}_{\geq 0}^n$ and $k \in [n]$. Then $b_k^{\downarrow(i)}, i \in [\mathcal{I}]$ is determinable as*

$$b_k^{\downarrow(i)} = \begin{cases} 0, & \text{if } i \neq \mathcal{I} \text{ and } b_k^{\downarrow(i+1)} \leq K \\ \left\lfloor b_k^{\downarrow}/2^{\mathcal{I}-i} - K \cdot \sum_{\ell=i}^{\mathcal{I}-1} \left(1/2^{\mathcal{I}-\ell}\right) \right\rfloor, & \text{otherwise.} \end{cases} \tag{3}$$

Proof Idea. We determine the interval in which $b_k^{\downarrow(i)}$ lies. The interval size is less than one. With Lemma 6, this implies the lemma. ◀

We now focus on the case, when $b_k^{\downarrow(i+1)}$ is large. Since we reduce the lower RHS in each entry by at most K such that the remaining entries are even, the remaining lower RHS can be divided by two. However, this does not directly imply that we can split the complete ILP into two equal sub-ILPs, as the upper RHS might have odd entries. To provide this, we give a procedure on how to reduce the solution of the n-fold within the iterations. We show that after the reduction, there exists a feasible solution for the remaining problem, which contains only even entries, which then implies that the complete RHS is even. Again, we make use of the support theorem (Lemma 5) of Eisenbrand and Shmonin [12, Theorem 1 (ii)]. Define

$$\tilde{x}_k^{(i)} := \begin{cases} x_k^{(i)}, & \text{if } \left\|x_k^{(i)}\right\|_1 \leq K \\ \left(\tilde{x}_{k,1}^{(i)}, \ldots, \tilde{x}_{k,t_k}^{(i)}\right)^T, & \text{otherwise.} \end{cases} \tag{4}$$

In the latter case, determine $\left(\tilde{x}_{k,1}^{(i)}, \ldots, \tilde{x}_{k,t_k}^{(i)}\right)^T$ as follows (for details we refer to the full version [17], where we present a pseudocode description of this procedure).

1. Set $\tilde{x}_k^{(i)} = \mathbb{0}_{t_i}$.
2. For all odd entries $\ell \in [t_k]$ in $x_{k,\ell}^{(i)}$, add 1 to $\tilde{x}_{k,\ell}^{(i)}$. This ensures the correct parity.
3. Until $\|\tilde{x}_k^{(i)}\|_1$ equals the desired value K or $K-1$, add 2 to an arbitrary component $\tilde{x}_{k,\ell}^{(i)}$, with $x_{k,\ell}^{(i)} - \tilde{x}_{k,\ell}^{(i)} \geq 0$. Note that there always exists such entry as otherwise, we would have $b_k^{\downarrow(i)} \leq K$ and the RHS of the remaining problem would be $\mathbb{0}$.

This procedure ensures that $x^{(i)} - \tilde{x}^{(i)} =: \hat{x}^{(i)}$ only consists of even components. Therefore, the RHS $\hat{b}^{(i)}$ of such ILP $\mathcal{A}\hat{x}^{(i)} = \hat{b}^{(i)}$ does not have any odd entries. Therefore, the complete problem can be split into two equal sub-problems.

Using the previously described reduction strategies, we now deduce how the upper RHS $b^{\uparrow(i)}$ changes during an iteration. As we do not know the solution x of (♠), the intermediate solutions $x^{(i)}$ are also unknown. Note that the described algorithm only gives a procedure to derive the current intermediate solution, assuming the prior one is known. Therefore, we cannot uniquely determine the upper RHS of the sub-problems. In the following, we often refer to a feasible RHS b as a feasible upper RHS or point $b^{\uparrow}$ when $b^{\downarrow}$ is apparent from the context. Recall that it is enough to determine the feasible points near ${}^{b^{\uparrow}}/_{2^{\mathcal{I}-i}}$ for all $i \in [\mathcal{I}]$. Now, we determine the size of the boxes in which a feasible point must lie if (♠) is feasible. For this purpose set $D := n \cdot K \cdot \Delta$. The next lemma states that for given small lower RHS $\tilde{b}^{\downarrow(i)}$ the size of the upper RHS $\tilde{b}^{\uparrow(i)}$ is bounded by D in each dimension (Figure 2).

▶ **Lemma 8** (✂). *Let $\tilde{b}^{(i)} = \left(\tilde{b}^{\uparrow(i)}, \tilde{b}^{\downarrow(i)}\right)^T$ with $i \in [\mathcal{I}], \|\tilde{b}^{\downarrow(i)}\|_\infty \leq K$ and assume $\mathcal{A}\tilde{x}^{(i)} = \tilde{b}^{(i)}$ is feasible. Then it holds that $\left\|\tilde{b}^{\uparrow(i)}\right\|_\infty \leq D$.*

With this property, we can derive the boxes with all relevant feasible upper RHS of the sub-problems. The center of the box in iteration i is ${}^{b^\uparrow}/_{2^{\mathcal{I}-i}}$ and the side length in each dimension is $2D$ (Figure 2).

▶ **Lemma 9** (✂). *Let $b = (b^\uparrow, b^\downarrow)^T$ and assume $\mathcal{A}x = b$ is feasible. Then the following two properties hold:*

(i) *For all $i \in \{2, \dots, \mathcal{I}\}$, there exists $b^{\uparrow(i)} \in \mathbb{Z}^r_{\geq 0}$ with $\left\| {}^{b^\uparrow}/_{2^{\mathcal{I}-i}} - b^{\uparrow(i)} \right\|_\infty \leq D$ and $\mathcal{A}x^{(i)} = (b^{\uparrow(i)}, b^{\downarrow(i)})^T$ is feasible.*

(ii) *There exists $b^{\uparrow(1)} \in \mathbb{Z}^r_{\geq 0}$ with $\|b^{\uparrow(1)}\|_\infty \leq D$ and $\mathcal{A}x^{(1)} = (b^{\uparrow(1)}, b^{\downarrow(1)})^T$.*

This means that if there exists a feasible point $b^{\uparrow(i)}$ near ${}^{b^\uparrow}/_{2^{\mathcal{I}-i}}$, there also exists another feasible point $b^{\uparrow(i-1)}$ near ${}^{b^\uparrow}/_{2^{\mathcal{I}-(i-1)}}$ for all iterations $i \in \{3, \dots, \mathcal{I}\}$. We also find a feasible point $b^{\uparrow(1)}$ near $\mathbb{0}$ if there is a feasible point $b^{\uparrow(2)}$ near ${}^{b^\uparrow}/_{2^{\mathcal{I}-2}}$. Altogether, this gives us a path from $\mathbb{0}$ to n, with at least one feasible point near each ${}^{b^\uparrow}/_{2^{\mathcal{I}-i}}$ for all $i \in \{2, \dots, \mathcal{I}\}$.

Now, we determine the total number of needed iterations by showing how to determine the number of iterations until the lower RHS equals $\mathbb{0}$. Let $\mathcal{I}_k$ be the number of *non-zero* iterations of machine-type k. An iteration i of machine-type k is non-zero when $b_k^{\downarrow(i)} \neq 0$.

▶ **Lemma 10** (✂). *Let $k \in [n]$, $b_k^\downarrow \in \mathbb{Z}_{\geq 0}$. Then the number of non-zero iterations $\mathcal{I}_k$ can be determined by*

$$\mathcal{I}_k = \begin{cases} \log\left({}^{(b_k^\downarrow + K)}/_{(2K+1)}\right) + 2, & \text{if } \log\left({}^{(b_k^\downarrow + K)}/_{(2K+1)}\right) \text{ is integer} \\ \left\lceil \log\left({}^{(b_k^\downarrow + K)}/_{(2K+1)}\right) \right\rceil + 1, & \text{otherwise.} \end{cases}$$

Proof Idea. We use (3) in Lemma 7 to determine the iteration i with $b_k^{\downarrow(i)} > 0$ and $b_k^{\downarrow(i-1)} = 0$. As calculating $b_k^{\downarrow(i)}$ is deterministic, this can be calculated uniquely. ◀

Having determined the number of non-zero iterations of each machine-type, we can directly derive the total number of iterations $\mathcal{I}$, as $\mathcal{I} = \max_{k \in [n]} \mathcal{I}_k$.

▶ **Lemma 11** (✂). *For the total number of iterations $\mathcal{I}$ it holds that*

$$\mathcal{I} = \begin{cases} \log\left({}^{(b_{\max}^\downarrow + K)}/_{(2K+1)}\right) + 2, & \text{if } \log\left({}^{(b_{\max}^\downarrow + K)}/_{(2K+1)}\right) \text{ is integer} \\ \left\lceil \log\left({}^{(b_{\max}^\downarrow + K)}/_{(2K+1)}\right) \right\rceil + 1, & \text{otherwise.} \end{cases}$$

With this in mind, we are now able to derive an algorithm where we start with a small n-fold ILP in the first iteration and build up the overall solution of (♠) step by step.

3.2 Solving the n-fold ILP

In this section, we introduce an algorithm that determines the feasibility of the n-fold ILP (♠) with only non-negative entries in $\mathcal{A}$. Combined with the reduction in Section 3.3, this results in the following:

▶ **Lemma 12.** *The feasibility of combinatorial n-fold ILPs (♠) where $\mathcal{A}$ contains only non-negative coefficients and Δ is the largest coefficient in $\mathcal{A}$ can be determined in time $(nr\Delta)^{O(r)} \log(b_{def})$.*

Proof Idea. In the preceding section we derived the property that we only have to solve small sub-problems (Lemma 8) and that we can combine their solutions in an efficient way to solve (♠). We split the algorithm into two parts. In the preprocessing we generate solutions for all required small sub-problems. Then, we generate the final solution by combining the intermediate solutions. A detailed proof of both the dynamic program and the combining step can be found in the full version [17].

Preprocessing. First, determine the number of non-zero iterations $\mathcal{I}_k$ for all $k \in [n]$ and the total number of iterations $\mathcal{I}$ with Lemma 10 and Lemma 11. Then determine all required lower RHS $b^{\downarrow(i)}, \tilde{b}^{\downarrow(i)}$ and $\hat{b}^{\downarrow(i)}$ for all $i \in [\mathcal{I}]$ as described in Section 3.1. For each vector $\tilde{b}^{\downarrow(i)}$ define the set of required upper RHS in iteration i :

$$\tilde{N}^{(i)} = \left\{\tilde{b}^{\uparrow(i)} \in \mathbb{Z}^r_{\geq 0} \mid \mathcal{A}\tilde{x}^{(i)} = \left(\tilde{b}^{\uparrow(i)}, \tilde{b}^{\downarrow(i)}\right)^T, \tilde{x}^{(i)} \in \mathbb{Z}^h_{\geq 0}, \tilde{b}^{\uparrow(i)} \leq D\right\}.$$

The elements in $\tilde{N}^{(i)}$ can be calculated via dynamic programming (Algorithm 1). Instead of determining the feasibility of the complete n-fold $\mathcal{A}\tilde{x}^{(i)} = \left(\tilde{b}^{\uparrow(i)}, \tilde{b}^{\downarrow(i)}\right)^T$, we split it into its bricks. For each brick $k \in [n]$ and upper RHS $\nu \in \{0, \ldots, K\Delta\}^r$, we determine the feasibility of $A_k\tilde{x}_k = \nu$ with $\|\tilde{x}_k\|_1 = \tilde{b}_k^{\downarrow(i)}$. These feasible points are then combined into the set $\tilde{N}^{(i)}$.

Algorithm 1 dynamicProgram. The base table (BT) contains the feasibility of all possible points for each individual block. The dynamic table DT combines the feasibility of the sub-problems. The set of required points is returned.

Input: Lower RHS $\tilde{b}^{\downarrow(i)}$
Output: Set $\tilde{N}^{(i)}$ of required points (upper RHS)

1: **for** all $k \in [n]$ and $\nu \in \{0, \ldots, K\Delta\}^r$ **do** ▷ Base Case

2: $$BT(\nu, k) \leftarrow \begin{cases} \texttt{true}, & \text{if } A_k\tilde{x}_k = \nu \text{ with } \|\tilde{x}_k\|_1 = \tilde{b}_k^{\downarrow(i)}, \tilde{x}_k \in \mathbb{Z}^{t_i}_{\geq 0} \text{ is feasible} \\ \texttt{false}, & \text{otherwise} \end{cases}$$

3: $DT(\nu, 1) \leftarrow BT(\nu, 1)$

4: **for** all $k = 2, \ldots, n$ and $\nu \in \{0, \ldots, D\}^r$ **do** ▷ Inductive Step

5: $$DT(\nu, k) \leftarrow \bigvee_{\substack{\nu'+\nu''=\nu, \\ \nu', \nu'' \text{ integral vectors}}} \left(DT(\nu', k-1) \wedge BT(\nu'', k)\right)$$

return $\left\{\nu \in \{0, \ldots, D\}^r \mid DT(\nu, n) = \texttt{true}\right\}$

The feasibility of the small sub-problems (base case) can be determined using a standard ILP algorithm. Applying the Jansen-Rohwedder algorithm [21] yields a running time of

$$O(\sqrt{r+1}\Delta)^{(1+o(1))(r+1)} + O(t_k(r+1)). \tag{5}$$

Combining the Solutions. Having solved all small sub-problems, we are now able to iteratively derive the feasibility of (♠). In the first iteration, set $N^{(1)} := \tilde{N}^{(1)}$ as the set of feasible points after this iteration. Note that for all $b^{\uparrow(1)} \in N^{(1)}$ it holds that $\mathcal{A}x^{(1)} = (b^{\uparrow(1)}, b^{\downarrow(1)})^T$ is feasible.

In every other iteration $i = 2, \ldots, \mathcal{I}$, we first double the solutions of the previous iteration and obtain the feasible ILP $\mathcal{A}\hat{x}^{(i)} = (\hat{b}^{\uparrow(i)}, \hat{b}^{\downarrow(i)})^T$, with $\hat{x}^{(i)} = 2x^{(i-1)}, \hat{b}^{\downarrow(i)} = 2b^{\downarrow(i-1)}$ and $\hat{b}^{\uparrow(i)} = 2b^{\uparrow(i-1)}$. Let $\hat{N}^{(i)} := \left\{\hat{b}^{\uparrow(i)} \in \mathbb{Z}^r_{\geq 0} \mid \hat{b}^{\uparrow(i)} = 2b^{\uparrow(i-1)}, b^{\uparrow(i-1)} \in N^{(i-1)}\right\}$ be the set of feasible points after that step. We now construct the set $N^{(i)}$ of the feasible points after this iteration by combining all vectors in $\hat{N}^{(i)}$ with all vectors in $\tilde{N}^{(i)}$ and only keep the ones close enough to $b^{\uparrow}/2^{\mathcal{I}-i}$ (Lemma 9), i.e.,

$$N^{(i)} := \left\{b^{\uparrow(i)} \in \mathbb{Z}^r_{\geq 0} \mid b^{\uparrow(i)} = \hat{b}^{\uparrow(i)} + \tilde{b}^{\uparrow(i)}, \hat{b}^{\uparrow(i)} \in \hat{N}^{(i)}, \tilde{b}^{\uparrow(i)} \in \tilde{N}^{(i)}, \left\|\frac{b^{\uparrow}}{2^{\mathcal{I}-i}} - b^{\uparrow(i)}\right\|_\infty \leq D\right\}.$$

Note again that for all $b^{\uparrow(i)} \in N^{(i)}$ it holds that $\mathcal{A}x^{(i)} = (b^{\uparrow(i)}, b^{\downarrow(i)})^T$ is feasible.

Repeating these steps results in a set $N^{(\mathcal{I})}$ of feasible points with $\mathcal{A}x^{(\mathcal{I})} = \left(b^{\uparrow(\mathcal{I})}, b^{\downarrow(\mathcal{I})}\right)^T$ and $b^{\downarrow(\mathcal{I})} = b^{\downarrow}, b^{\uparrow(\mathcal{I})} \in N^{(\mathcal{I})}$. Therefore, if $b^{\uparrow} \in N^{(\mathcal{I})}$ the n-fold ILP (♠) is feasible.

Running Time. The running time of the preprocessing is dominated by the dynamic program. In the worst case, it is called $O(n)$ times. Creating the base table (BT) of the DP, we have to solve $n(K\Delta+1)^r$ small sub-problems, each having a running time of (5). The dynamic program then constructs a dynamic table (DT) with $n(D+1)^r$ entries. Each can be calculated by doing a boolean convolution using FFT which takes time $O(n(D+1)^r \cdot \log(n(D+1)^r))$ (see e.g. [5]). With $t_k \leq h$ for all $k \in [n]$, the total running time for the preprocessing amounts to

$$\underbrace{n(K\Delta+1)^r \cdot O(\sqrt{r+1}\Delta)^{(1+o(1))(r+1)} + O(h(r+1))}_{\text{Base Case}} + \underbrace{n^2(D+1)^{O(r)} \cdot \log(n(D+1)^r)}_{\text{Inductive Step}}$$
$$= (D+1)^{O(r)}$$

The number of iterations, we need to combine the solutions is generally bounded by $\log(b^{\downarrow}_{\max})$. However, if the upper RHS is relatively small in all entries, it becomes $\mathbb{0}$ before $b^{\downarrow}$ does. In order to be feasible in that case, we only need to check whether a $\mathbb{0}$ in each A_i where the corresponding entry in the lower RHS is not 0, exists. Thus, these sub-problems are trivial to solve and therefore, we may say that we need $\log(b_{\text{def}})$ iterations, where $b_{\text{def}} := \min(b^{\uparrow}_{\max}, b^{\downarrow}_{\max})$. Each set $N^{(i)}, \tilde{N}^{(i)}$ and $\hat{N}^{(i)}$ contains at most $(D+1)^r$ vectors. Therefore, determining each set is possible in time $D^{O(r)}$. With $h \leq (\Delta+1)^r$ this yields a total running time of $(nr\Delta)^{O(r)} \log(b_{\text{def}})$. ◀

3.3 Allowing Negative Coefficients

In this section, we provide a linear-time reduction from a general combinatorial n-fold ILP (♠) to an ILP of the same form which only allows non-negative entries in $\mathcal{A}$. More concretely, we prove:

▶ **Lemma 13.** *Let $X \subseteq \mathbb{Z}^h_{\geq 0}$ be the set of feasible solutions of the n-fold ILP (♠) with sub-matrices $A_i \in \mathbb{Z}^{r \times t_i} \forall i \in [n]$ and RHS vector $b \in \mathbb{Z}^{r+n}$ and let Δ be the largest absolute value in A_i, i.e., $\Delta = \max_{i \in [n]} \|A_i\|_\infty$. Then there are matrices $A'_i \in \mathbb{Z}^{r \times t_i}_{\geq 0} \forall i \in [n]$ and a vector $b' \in \mathbb{Z}^{r+n}_{\geq 0}$ such that X is the set of feasible solutions to $\mathcal{A}'x = b'$ of the form (♠) with sub-matrices A_i and the entries in A'_i are bounded by $\Delta' = 2\Delta$.*

Proof. Let $i \in [n]$. We construct A'_i as by adding Δ to each entry in A_i, i.e., $A'_i[k,\ell] := A_i[k,\ell] + \Delta$, where $A_i[k,\ell]$ is the entry in A_i that is in the k-th row and ℓ-th column (analogous notation for $A'_i[k,\ell]$). Since the absolute entries in A_i are smaller than or equal to Δ, all entries in A'_i are non-negative and upper bounded by $2\Delta =: \Delta'$.

Now we show how the RHS is adapted. Note that the local constraints of (♠) define the 1-norm of the solution vectors, i.e., for all $x \in X$ it holds that $\|x\|_1 = \|b^{\downarrow}\|_1$. Viewing the matrix-multiplication as summing up vectors (columns of the matrix), we are summing up exactly $\|x\|_1$ vectors. Therefore, by changing the constraint matrix as described above, we know that exactly $\|b^{\downarrow}\|_1 \cdot \Delta$ are added to each entry of the upper RHS. Thus, we define $b^{\uparrow'}$ as follows: For each $j \in [r]$ set $b^{\uparrow'}_j := b^{\uparrow} + \|b^{\downarrow}\|_1 \cdot \Delta$. Since the local constraints do not change, the lower RHS stays the same that is $b^{\downarrow'} := b^{\downarrow}$. ◀

3.4 Solving the Optimization Problem

While we focused our discussions on the problem of deciding whether a feasible solution to a given combinatorial n-fold exists, we note that some modifications further extend our results to finding an optimal solution according to some objective function, for instance $\max c^T x, c \in \mathbb{Z}^h$. We need to adapt the upper bound on the support (Lemma 5) and the dynamic program (Algorithm 1) according to this new objective function. For $\max c^T x, c \in \mathbb{Z}^h$, the running time of the dynamic program changes as follows: Instead of determining the feasibility of the small sub-problems in the base case, we need the optimal solution of those. The currently best algorithm by Jansen and Rohwedder [21] achieves a running time of $O(\sqrt{r+1}\Delta)^{2(r+1)} + O(h(r+1))$. In the inductive step, we replace the boolean convolution with $(\max, +)$-convolution, which has generally has quadratic running time. This and the change of the support bound to $2(r+2)(\log(r+2)+\Delta+2)$ does not affect the asymptotic running time which yields:

▶ **Theorem 1.** *Combinatorial n-fold ILPs (♠) with respect to the objective of maximizing $c^T x, c \in \mathbb{Z}^h_{\geq 0}$ and where Δ is the largest absolute value in $\mathcal{A}$ can be solved in time $(nr\Delta)^{O(r)} \log(b_{def})$.*

4 Applying the Algorithm

To complete the discussion of our algorithm, we present exemplary applications of the algorithm to different contexts. Here, we introduce the considered problems, while we refer to the full version [17] for the detailed applications and resulting running times.

Scheduling on Uniform Machines

A scheduling instance is defined by the number of jobs $n \in \mathbb{Z}^d_{\geq 0}$ with processing times $p \in \mathbb{Z}^d_{\geq 0}$ and the number of machines $m \in \mathbb{Z}^\tau_{\geq 0}$ with speed values $s \in \mathbb{Z}^\tau_{\geq 1}$. This means that $N := \|n\|_1$ jobs of d different sizes have to be scheduled on $M := \|m\|_1$ machines with τ_I different speeds. Since the processing times are integer, the total number of different job-types d is bounded by $d \leq p_{\max}$. Each job has to be executed on a single machine.

Let $\mathcal{M}$ be the set of all machines. A *schedule* $\sigma \colon \mathbb{Z}^d_{\geq 0} \to \mathcal{M}$ is a mapping that assigns all jobs to the available machines. The *load* $L(m^{(k)})^{(\sigma)}$ of machine $m^{(k)} \in \mathcal{M}$ is the sum of the sizes of all jobs assigned to that machine by the schedule σ. Assume $m^{(k)}$ has speed s_k. Then we define the *completion time* of machine $m^{(i)}$ by $C(m^{(i)})^{(\sigma)} := {}^{L(m^{(i)})^{(\sigma)}}/_{s_k}$, i.e., the time when machine $m^{(i)}$ has finished processing all jobs that were assigned to it by σ.

The objective functions that we consider in this work are:

- **Makespan Minimization ($C_{\max}$):** $\min_\sigma \left(\max_{m^{(i)} \in \mathcal{M}} C(m^{(i)})^{(\sigma)} \right)$
- **Santa Claus ($C_{\min}$):** $\max_\sigma \left(\min_{m^{(i)} \in \mathcal{M}} C(m^{(i)})^{(\sigma)} \right)$.

Closest String Problem

When considering the Closest String problem, we are given a set of k strings $S = \{s_1, \ldots, s_k\}$ of length $L \in \mathbb{Z}_{\geq 1}$ from alphabet $[k]$ and an integer $d \in \mathbb{Z}_{\geq 0}$. The task is to find a string $c \in [k]^L$ of length L (called the "closest string") such that the Hamming distance $d_H(c, s_i)$ between c and any string $s_i \in S$ is at most d. The Hamming distance for two equal-length strings is the number of positions at which they differ.

Imbalance Problem

The input to the Imbalance problem is an undirected graph $G = (V, E)$ with n vertices. An ordering of the vertices in G is a bijective function $\pi = V \to [n]$. For $v \in V$, we define neighborhood of v by $N(v) := \{u \in V \mid (u, v) \in E\}$, the set of left neighbors $L(v) := \{u \in N(v) \mid \pi(u) < \pi(v)\}$ and the set of right neighbors $R(v) := \{u \in N(v) \mid \pi(v) < \pi(u)\}$. Note that $R(v) = N(v) \setminus L(v)$. The imbalance of at $v \in V$ is defined as $\iota_\pi(v) = ||R(v)| - |L(v)||$ and the total imbalance of the ordering π is $\iota(\pi) = \sum_{v \in V} \iota_\pi(v)$. The aim is now to find an ordering π that minimizes the total imbalance.

5 Conclusion

We have shown that we can solve combinatorial n-fold ILPs in time $(nr\Delta)^{O(r)} \log(b_{\text{def}})$. Our algorithm builds on insights regarding the partition of the entire problem into smaller and smaller sub-problems, which we solve and use to reconstruct the entire problem in a novel manner. Complementing our algorithmic results, we present applications of our algorithm to the world of scheduling on uniform machines, the closest string problem and imbalance. In the case of scheduling on uniform machines, we can apply techniques to bound Δ and achieve an algorithm with a running time matching a lower bound provided by the ETH for both makespan minimization and Santa Claus. For closest string our algorithm matches the current state of the art, but can, in contrast to existing algorithms, leverage bounded number of column-types to achieve improved running times. Finally, when applied to imbalance, our algorithm results in an improved running time over the state-of-the-art. For both closest string and imbalance our algorithm matches the current state of the art, but can, in contrast to existing algorithms, leverage bounded number of column/vertex types to achieve improved running times.

References

1 Matthias Aschenbrenner and Raymond Hemmecke. Finiteness theorems in stochastic integer programming. *Foundations of Computational Mathematics*, 7(2):183–227, 2007. doi:10.1007/S10208-005-0174-1.

2 Olav Røthe Bakken. Arrangement problems parameterized by neighbourhood diversity. *Master's thesis, University of Bergen*, 2018.

3 Sebastian Berndt, Hauke Brinkop, Klaus Jansen, Matthias Mnich, and Tobias Stamm. New support size bounds for integer programming, applied to makespan minimization on uniformly related machines. In *International Symposium on Algorithms and Computation (ISAAC)*, volume 283 of *LIPIcs*, pages 13:1–13:18, 2023. doi:10.4230/LIPICS.ISAAC.2023.13.

4 Therese Biedl, Timothy M. Chan, Yashar Ganjali, Mohammad Taghi Hajiaghayi, and David R. Wood. Balanced vertex-orderings of graphs. *Discrete Applied Mathematics*, 148(1):27–48, 2005. doi:10.1016/J.DAM.2004.12.001.

5 Karl Bringmann and Vasileios Nakos. Fast n-fold boolean convolution via additive combinatorics. In *International Colloquium on Automata, Languages, and Programming, (ICALP)*, volume 198 of *LIPIcs*, pages 41:1–41:17, 2021. doi:10.4230/LIPICS.ICALP.2021.41.

6 Lin Chen, Klaus Jansen, and Guochuan Zhang. On the optimality of exact and approximation algorithms for scheduling problems. *Journal of Computer and System Sciences*, 96:1–32, 2018. doi:10.1016/J.JCSS.2018.03.005.

7 Lin Chen, Dániel Marx, Deshi Ye, and Guochuan Zhang. Parameterized and approximation results for scheduling with a low rank processing time matrix. In *Symposium on Theoretical Aspects of Computer Science, (STACS)*, volume 66 of *LIPIcs*, pages 22:1–22:14, 2017. doi:10.4230/LIPICS.STACS.2017.22.

8 Jana Cslovjecsek, Friedrich Eisenbrand, Christoph Hunkenschröder, Lars Rohwedder, and Robert Weismantel. Block-structured integer and linear programming in strongly polynomial and near linear time. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1666–1681, 2021. doi:10.1137/1.9781611976465.101.

9 Jana Cslovjecsek, Friedrich Eisenbrand, Michal Pilipczuk, Moritz Venzin, and Robert Weismantel. Efficient sequential and parallel algorithms for multistage stochastic integer programming using proximity. In *European Symposium on Algorithms, ESA*, volume 204 of *LIPIcs*, pages 33:1–33:14, 2021. doi:10.4230/LIPICS.ESA.2021.33.

10 Friedrich Eisenbrand, Christoph Hunkenschröder, and Kim-Manuel Klein. Faster algorithms for integer programs with block structure. In *International Colloquium on Automata, Languages, and Programming, (ICALP)*, volume 107 of *LIPIcs*, pages 49:1–49:13, 2018. doi:10.4230/LIPICS.ICALP.2018.49.

11 Friedrich Eisenbrand, Christoph Hunkenschröder, Kim-Manuel Klein, Martin Koutecký, Asaf Levin, and Shmuel Onn. An algorithmic theory of integer programming, 2022. arXiv:1904.01361.

12 Friedrich Eisenbrand and Gennady Shmonin. Carathéodory bounds for integer cones. *Operations Research Letters*, 34(5):564–568, 2006. doi:10.1016/J.ORL.2005.09.008.

13 Michael R. Fellows, Daniel Lokshtanov, Neeldhara Misra, Frances A. Rosamond, and Saket Saurabh. Graph layout problems parameterized by vertex cover. In *International Symposium on Algorithms and Computation (ISAAC)*, volume 5369 of *LNCS*, pages 294–305, 2008. doi:10.1007/978-3-540-92182-0_28.

14 Tomas Gavenciak, Martin Koutecký, and Dusan Knop. Integer programming in parameterized complexity: Five miniatures. *Discrete Optimization*, 44(Part):100596, 2022. doi:10.1016/J.DISOPT.2020.100596.

15 Jens Gramm, Rolf Niedermeier, and Peter Rossmanith. Fixed-parameter algorithms for CLOSEST STRING and related problems. *Algorithmica*, 37(1):25–42, 2003. doi:10.1007/S00453-003-1028-3.

16 Raymond Hemmecke, Shmuel Onn, and Lyubov Romanchuk. n-fold integer programming in cubic time. *Mathematical Programming*, 137(1-2):325–341, 2013. doi:10.1007/S10107-011-0490-Y.

17 Klaus Jansen, Kai Kahler, Lis Pirotton, and Malte Tutas. New algorithm for combinatorial n-folds and applications, 2025. arXiv:2409.04212.

18 Klaus Jansen, Kai Kahler, and Esther Zwanger. Exact and approximate high-multiplicity scheduling on identical machines. In *Conference on Algorithms and Complexity (CIAC)*, volume 15679 of *LNCS*, pages 1–17, 2025. doi:10.1007/978-3-031-92932-8_1.

19 Klaus Jansen, Kim-Manuel Klein, Marten Maack, and Malin Rau. Empowering the configuration-IP: new PTAS results for scheduling with setup times. *Mathematical Programming*, 195(1):367–401, 2022. doi:10.1007/S10107-021-01694-3.

20 Klaus Jansen, Alexandra Lassota, and Lars Rohwedder. Near-linear time algorithm for n-fold ILPs via color coding. *SIAM Journal on Discrete Mathematics*, 34(4):2282–2299, 2020. doi:10.1137/19M1303873.

21 Klaus Jansen and Lars Rohwedder. On integer programming, discrepancy, and convolution. *Mathematics of Operations Research*, 48(3):1481–1495, 2023. doi:10.1287/MOOR.2022.1308.

22 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.

23 Jan Kára, Jan Kratochvíl, and David R. Wood. On the complexity of the balanced vertex ordering problem. *Discrete Mathematics & Theoretical Computer Science*, 9(1), 2007. doi:10.46298/DMTCS.383.

24 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger, editors, *Complexity of Computer Computations*, pages 85–103. Springer US, Boston, MA, 1972. doi:10.1007/978-1-4684-2001-2_9.

25 Dusan Knop and Martin Koutecký. Scheduling meets n-fold integer programming. *Journal of Scheduling*, 21(5):493–503, 2018. doi:10.1007/S10951-017-0550-0.

26 Dusan Knop, Martin Koutecký, Asaf Levin, Matthias Mnich, and Shmuel Onn. Parameterized complexity of configuration integer programs. *Operations Research Letters*, 49(6):908–913, 2021. doi:10.1016/J.ORL.2021.11.005.

27 Dusan Knop, Martin Koutecký, Asaf Levin, Matthias Mnich, and Shmuel Onn. High-multiplicity n-fold IP via configuration LP. *Mathematical Programming*, 200(1):199–227, 2023. doi:10.1007/S10107-022-01882-9.

28 Dusan Knop, Martin Koutecký, and Matthias Mnich. Combinatorial n-fold integer programming and applications. *Mathematical Programming*, 184(1):1–34, 2020. doi:10.1007/S10107-019-01402-2.

29 Dušan Knop and Martin Koutecký. Scheduling Meets N-Fold Integer Programming. *Journal of Scheduling*, 21(5):493–503, October 2018. doi:10.1007/S10951-017-0550-0.

30 Martin Koutecký, Asaf Levin, and Shmuel Onn. A parameterized strongly polynomial algorithm for block structured integer programs. In *International Colloquium on Automata, Languages, and Programming, (ICALP)*, volume 107 of *LIPIcs*, pages 85:1–85:14, 2018. doi:10.4230/LIPICS.ICALP.2018.85.

31 Martin Koutecký and Johannes Zink. Complexity of scheduling few types of jobs on related and unrelated machines. *Journal of Scheduling*, 28(1):139–156, 2025. doi:10.1007/S10951-024-00827-8.

32 Jesús A. De Loera, Raymond Hemmecke, Shmuel Onn, and Robert Weismantel. N-fold integer programming. *Discrete Optimization*, 5(2):231–241, 2008. doi:10.1016/J.DISOPT.2006.06.006.

33 Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Imbalance is fixed parameter tractable. *Information Processing Letters*, 113(19-21):714–718, 2013. doi:10.1016/J.IPL.2013.06.010.

34 Neeldhara Misra and Harshil Mittal. Imbalance parameterized by twin cover revisited. *Theoretical Computer Science*, 895:1–15, 2021. doi:10.1016/J.TCS.2021.09.017.

35 Christos H. Papadimitriou. On the complexity of integer programming. *Journal of the ACM*, 28(4):765–768, 1981. doi:10.1145/322276.322287.

36 Victor Reis and Thomas Rothvoss. The subspace flatness conjecture and faster integer programming. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 974–988, 2023. doi:10.1109/FOCS57990.2023.00060.

37 Lars Rohwedder. Eth-tight FPT algorithm for makespan minimization on uniform machines. In *International Colloquium on Automata, Languages, and Programming, (ICALP)*, volume 334 of *LIPIcs*, pages 126:1–126:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPICS.ICALP.2025.126.

38 Lars Rohwedder and Karol Wegrzycki. Fine-grained equivalence for problems related to integer linear programming. In *Innovations in Theoretical Computer Science (ITCS)*, volume 325 of *LIPIcs*, pages 83:1–83:18, 2025. doi:10.4230/LIPICS.ITCS.2025.83.

Designing Compact ILPs via Fast Witness Verification

Michał Włodarczyk
University of Warsaw, Poland

Abstract

The standard formalization of preprocessing in parameterized complexity is given by kernelization. In this work, we depart from this paradigm and study a different type of preprocessing for problems without polynomial kernels, still aiming at producing instances that are easily solvable in practice. Specifically, we ask for which parameterized problems an instance (I, k) can be reduced in polynomial time to an integer linear program (ILP) with $\mathsf{poly}(k)$ constraints.

We show that this property coincides with the parameterized complexity class WK[1], previously studied in the context of Turing kernelization lower bounds. In turn, the class WK[1] enjoys an elegant characterization in terms of witness verification protocols: a yes-instance should admit a witness of size $\mathsf{poly}(k)$ that can be verified in time $\mathsf{poly}(k)$. By combining known data structures with new ideas, we design such protocols for several problems, such as r-Way Cut, Vertex Multiway Cut, Steiner Tree, and Minimum Common String Partition, thus showing that they can be modeled by compact ILPs. We also present explicit ILP and MILP formulations for Weighted Vertex Cover on graphs with small (unweighted) vertex cover number. We believe that these results will provide a background for a systematic study of ILP-oriented preprocessing procedures for parameterized problems.

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

Keywords and phrases integer programming, kernelization, nondeterminism, multiway cut

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.16

Related Version *Full Version*: https://arxiv.org/abs/2509.25445

Funding Supported by Polish National Science Centre SONATA-19 grant 2023/51/D/ST6/00155.

1 Introduction

Kernelization [38] provides a rigorous theory for data preprocessing in the following sense: given an instance I with parameter k we would like to transform it in polynomial time into an equivalent instance (I', k') so that $|I'| + k' \leq f(k)$ for a slowly growing function f. Historically, kernelization was meant as just another tool for establishing fixed-parameterized tractability [25] but later it grew into a mature theory on its own, especially after a hardness framework has been established [6]. The main question in this paradigm is: which parameterized problems admit polynomial kernels, that is, when can the function f be estimated as polynomial? From the practical point of view, once the instance size is reduced, it may become amenable to brute-force or heuristic methods.

In recent years, empirical observations have provided evidence that analyzing preprocessing through the lens of kernelization might be too restrictive. In the annual PACE challenge [29, 43, 53, 54], co-organized with the IPEC conference, competitors solve a collection of instances of a certain NP-hard problem related to parameterized complexity. It turned out that the many successful submissions [3, 4, 9, 23, 24] do not follow the algorithms from FPT handbooks but instead reduce the problem to (mixed) integer linear programming (ILP) and then employ a specialized solver. Even though the contestants are not allowed to use

© Michał Włodarczyk;
licensed under Creative Commons License CC-BY 4.0

the state-of-the-art industrial solvers, already the open source solvers often outperform sophisticated FPT algorithms with provable worst-case guarantees. However, it is usually challenging to model the problem in question as a concise ILP.

In our opinion, parameterized complexity should seek an explanation of this phenomenon and understand which problems can be handled with ILP-driven preprocessing. In this work, we make a first step in this direction and investigate one of the possible ways of formalizing such a notion of preprocessing. Even though modeling NP-hard problems as succinct (mixed) ILPs has been a subject of operation research since the 1950s [14, 22, 37, 42, 65, 66, 71], no systematical study has addressed this question from the perspective of parameterized complexity so far.

What kind of preprocessing? Following the design of kernelization theory, we consider polynomial-time procedures that output an equivalent instance (of a possibly different decision problem) that should be well-structured as long as the initial parameter is not too large. While there are many ways to measure the difficulty of an ILP [1, 11, 17, 18, 28, 30, 40, 48, 36, 50], the most natural are the number of constraints and the number of variables. In this work, we focus on the first one.

Specifically, we are interested in processing an instance (I, k) in polynomial time to rewrite it as an ILP of the form $(Ax \leq b, x \in \mathbb{Z}^n_{\geq 0})$, where $A \in \mathbb{Z}^{m \times n}$, $b \in \mathbb{Z}^m$, and m is polynomially bounded in k. In other words, we would like to reduce solving the problem in question to feasibility check of an ILP with only $\mathsf{poly}(k)$ constraints. This strategy matches the assumption guiding the kernelization paradigm: when the parameter k is small, then heuristic algorithms should work fast in practice when the search space has "dimension"[1] $\mathsf{poly}(k)$. Naturally, it would be desirable to obtain ILP formulations with $m = \mathcal{O}(k)$ or $m = \mathcal{O}(k^2)$ but we believe that a new framework should first focus on establishing general bounds and more fine-grained results will follow. For comparison, the first work on polynomial kernelization for Feedback Vertex Set only guaranteed a kernel of size $\mathcal{O}(k^{11})$ [13] but it was soon improved to $\mathcal{O}(k^2)$ [69].

What kind of problems? Before we delve further into technicalities, let us reflect on for which parameterized problems the question above makes sense. First of all, we are interested in problems without (known) polynomial kernels. Secondly, any problem that can be reduced as described above must (1) belong to NP and (2) admit an FPT algorithm with running time of the form $2^{\mathsf{poly}(k)} \cdot n^{\mathcal{O}(1)}$ [51]. Restricting to only such problems allows us to assume $\log n \leq \mathsf{poly}(k)$ because otherwise the FPT running time above can be estimated as polynomial in n. Later we will also see another property that must be satisfied: an instance (I, k) should admit an NP-witness of size $\mathsf{poly}(k, \log |I|)$. Therefore, we should not hope to handle structural parameterizations like treewidth due to known barriers [27].

Fast witness verification. The question outlined above can be formalized in the following way: we ask which parameterized problems admit a *polynomial parameter transformation* (PPT) to ILP feasibility checking parameterized by the number of constraints. It is easy to see that Set Cover parameterized by the universe size admits such a reduction and we will present a reduction in the opposite direction as well. In turn, the class of problems reducible

[1] Usually the "dimension" of an ILP refers to the number of variables but the known FPT algorithms [32, 51] for solving ILP parameterized by the number of constraints m perform computations in a lattice of dimension m.

by PPT to Set Cover is known as WK[1]. The WK-hierarchy has been introduced by Hermelin, Kratsch, Sołtys, Wahlström, and Wu [45] in the context of Turing kernelization. What is interesting for us, the class WK[1] comprises those parameterized problems that admit an NP-witness of size $\mathsf{poly}(k)$ whose correctness can be verified in time $\mathsf{poly}(k)$. This elegant characterization has remained merely a curiosity because the main focus of the work [45] was to prove problem hardness with respect to the WK-hierarchy[2]. We shall exploit this characterization in a different context.

Related work. A more expressive variant of ILP allows one to write lower and upper bounds $\ell_i \leq x_i \leq u_i$ that do not count to the constraint limit (see Section 2). Rohwedder and Węgrzycki [68] considered such ILP parameterized by the number of constraints and studied fine-grained equivalences with problems such as Closest String, Discrepancy Minimization, and Set Multi-Cover. Kernelization of ILP Feasibility has been considered in [56, 59]. For more applications of ILP in parameterized complexity, see, e.g., [15, 41, 44, 55].

Apart from witness verification protocols studied here, there are other models of non-deterministic computation considered in parameterized complexity, mainly in the context of space limitations [7, 33, 67, 72].

Our contribution

We show that the class WK[1] provides an exact characterization of parameterized problems that can be reduced to ILP with few constraints as long as the entries in the corresponding matrix cannot be too large. The latter restriction is necessary because unbounded entries allow one to express Unbounded Subset Sum (which is NP-hard [62]) with just one equality constraint. More formally, we consider the feasibility problem of the form $(Ax \leq b,\ x \in \mathbb{Z}^n_{\geq 0})$, where $A \in \mathbb{Z}^{m \times n}$, $b \in \mathbb{Z}^m$, and define $\Delta(A)$ to be the maximal absolute value of an entry in the matrix A.

▶ **Theorem 1.** Integer Linear Programming Feasibility *is* WK[1]*-complete when:*

- *parameterized by* $m + \log(\Delta(A))$*,*
- *parameterized by* m *only, under the assumption that* $\Delta(A) = 1$ *and the vector* b *is given in unary.*

Consequently, a parameterized problem admits a polynomial parameter transformation to ILP Feasibility (under the considered parameterizations) if and only if it belongs to the class WK[1]. Notably, it follows that allowing for polynomially large numbers in an ILP does not lead to any qualitative advantage. A similar equivalence is known for the variant with bounded variables [68] but therein the reduction involves exponentially many variables. That variant is easily seen to be WK[1]-hard (Observation 13).

Hermelin et al. [45] already showed that WK[1] includes problems of local nature, such as k-Path, Connected Vertex Cover, Min Ones d-SAT, or Steiner Tree, all parameterized by the solution size. Motivated by Theorem 1, we show that this class captures many more problems, also those of seemingly non-local nature. To this end, we employ various data structures to design fast witness verification protocols. The definitions of the listed problems and the related results are discussed in the corresponding sections.

[2] This characterization is implicit in [45] so for completeness we provide a proof in the full version of the article.

▶ **Theorem 2.** *The following parameterized problems belong to the class* WK[1]*:*

- r-Way Cut */ solution size,*
- Vertex Multiway Cut */ solution size,*
- Minimum Common String Partition */ partition size,*
- Long Path */ feedback vertex set number,*
- Steiner Tree */ number of terminals,*
- Optimal Discretization */ solution size.*

Theorems 1 and 2 imply that all the listed problems can be reduced to ILP with $\mathsf{poly}(k)$ constraints. In fact, we provide parameterized reductions to Binary NDTM Halting, which then reduces to Set Cover, which in turn can be readily modeled by a compact ILP. We remark that the proofs of containment in WK[1] are not technically involved but they exploit dynamic data structures in a novel context. In our opinion, their value lies in creating a new link between the fields of integer programming, proof protocols, and data structures.

The results above only provide implicit reductions to ILP where the number constraints is an unspecified polynomial in the parameter. We complement those with an explicit reduction for Weighted Vertex Cover / VC, where the parameter is the vertex cover number of the input graph. We attain only linearly many constraints, however with a caveat that we require all the variables to be binary. In addition, we present a mixed integer linear program where the number of integral variables is also linear in the parameter.

▶ **Theorem 3.** Weighted Vertex Cover / VC *admits a linear parameter transformation to* Binary ILP Feasibility *parameterized by* $m + \log(\Delta(A))$*. Furthermore, the problem admits a linear parameter transformation to* MILP Feasibility *parametrized by the number of integral variables.*

Organization of the paper. In Section 2 we provide background on ILPs and MILPs, together with a discussion on the class WK[1]. Section 3 is devoted to proving Theorem 1 while the proof of Theorem 2 is given in Section 4. In Section 5 we give a closer look at Weighted Vertex Cover.

2 Preliminaries

On default, we measure the running time of algorithms in the standard RAM model with logarithmic word length. All the summoned algorithmic results are deterministic. Throughout the paper, we use the variable n to denote the input size during the reductions or the number of variables when analyzing integer programs. The number of constraints is denoted by m.

Integer linear programming. For $A \in \mathbb{Z}^{m \times n}$ and $b \in \mathbb{Z}^m$, we consider ILP in the *standard form without upper bounds*: $(Ax \leq b,\ x \in \mathbb{Z}^n_{\geq 0})$. Note that it is allowed to write inequalities "$\geq$" as well since one can multiply coefficients by -1. The problem Integer Linear Programming (ILP) Feasibility asks, given A, b, whether there exists a vector x satisfying the system above. One can also consider ILP in the *equality form*: $(Ax = b,\ x \in \mathbb{Z}^n_{\geq 0})$ where $A \in \mathbb{Z}^{m \times n}$ and $b \in \mathbb{Z}^m$. It is easy to transform a system of inequalities $(Ax \leq b,\ x \in \mathbb{Z}^n_{\geq 0})$ into the equality form by introducing m new non-negative variables and turning each inequality $\sum a_i x_i \leq b$ into equality $\sum a_i x_i + z = b$ where z is a fresh variable. This transformation does not affect the number of constraints and results in matrix A' obtained from A by inserting m columns with only 0 and 1 entries. We make note of this observation in the following terms. Recall that $\Delta(A)$ is the maximal absolute value of an entry in the matrix A.

▶ **Observation 4.** *A system of inequalities* ($Ax \leq b$, $x \in \mathbb{Z}^n_{\geq 0}$), $A \in \mathbb{Z}^{m \times n}$, $b \in \mathbb{Z}^m$, *can be transformed in polynomial time into an equivalent system of equalities* ($A'y = b$, $y \in \mathbb{Z}^{n+m}_{\geq 0}$) *where* $A' \in \mathbb{Z}^{m \times (n+m)}$ *and* $\Delta(A') = \max(1, \Delta(A))$.

Another common variant of ILP is the *standard form with upper bounds*: ($Ax \leq b$, $x \leq u$, $x \in \mathbb{Z}^n_{\geq 0}$) where $u \in \mathbb{Z}^n_{\geq 0}$. This form effectively expresses n additional inequalities which are not counted as constraints. When u is the all-1s-vector, we call the corresponding decision problem Binary ILP Feasibility. All variants admit FPT algorithms parameterized by the number of constraints m and $\Delta(A)$. On the other hand, parameterization by m alone is unlikely to be FPT even when the input is given in unary [39, Theorem 2].

▶ **Proposition 5** ([51, Theorem 5])**.** *Feasibility of an ILP in the standard form* **without** *upper bounds can be decided in time* $\mathcal{O}(m \cdot \Delta(A))^{2m} \cdot \log(||b||_\infty)$.

▶ **Proposition 6** ([32, Theorem 4.1])**.** *Feasibility of an ILP in the standard form* **with** *upper bounds can be decided in time* $(m \cdot \Delta(A))^{\mathcal{O}(m^2)} \cdot n + s^{\mathcal{O}(1)}$ *where* s *is the input encoding size.*

In *mixed integer linear programming* (MILP) we have two groups of variables: n integral ones and ℓ fractional ones. Let $x[a, b]$ denote the truncation of the vector x to coordinates $[a, b]$. In a variant with upper bounds we look for a vector x satisfying the system ($Ax \leq b$, $x \leq u$, $x[1, n] \in \mathbb{Z}^n_{\geq 0}$, $x[n+1, n+\ell] \in \mathbb{R}^\ell_{\geq 0}$) where $A \in \mathbb{Z}^{m \times (n+\ell)}$, $u \in \mathbb{Z}^{n+\ell}$, and $b \in \mathbb{Z}^m$. When the number n of integral variables is small, one can solve a MILP using an extension of Lenstra's algorithm. Note that in contrast to parameterization by m, now $\Delta(A)$ can be unbounded.

▶ **Proposition 7** ([52])**.** *Feasibility of a MILP with* n *integral variables can be determined in time* $n^{\mathcal{O}(n)} \cdot s^{\mathcal{O}(1)}$, *where* s *is the input encoding size.*

Class WK[1]. A *polynomial parameter transformation* (PPT) is a reduction between parameterized problems P, Q that runs in polynomial time and maps instance (I, k) of P to an equivalent instance (I', k') of Q satisfying $k' \leq \mathsf{poly}(k)$. The WK-hierarchy has been introduced as a hardness hierarchy for problems unlikely to admit a polynomial Turing kernel [45]. For each $t \geq 1$ the class WK[t] comprises problems that are PPT-reducible to Weighted SAT over a certain class of formulas, parameterized by the number of 1s in an assignment times $\log n$. Since we are only interested in the class WK[1], it is more convenient to rely on its equivalent characterization as the PPT-closure of the following WK[1]-complete problem [45].

Binary NDTM Halting	**Parameter:** k
Input: nondeterministic Turing machine M over a binary alphabet, integer k	
Question: can M halt on the empty string in k steps?	

Another canonical WK[1]-complete problem is Set Cover.

Set Cover	**Parameter:** $\lvert U \rvert$
Input: set U, family of sets $\mathcal{F} \subseteq 2^U$, integer ℓ	
Question: is there a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ of size ℓ such that the union of $\mathcal{F}'$ is U?	

WK[1]-completeness of Binary NDTM Halting gives rise to a particularly elegant characterization of WK[1] in terms of witness verification protocols.

▶ **Definition 8.** *A* witness verification protocol *for a parameterized problem L is a pair of algorithms $(\mathcal{A}, \mathcal{B})$ with the following specification.*

Given an instance (I, k) the algorithm $\mathcal{A}$ runs in polynomial time and outputs a string $s(I,k) \in \{0,1\}^*$ and integer $\ell(I,k)$. The algorithm $\mathcal{B}$ is given the concatenation of $s(I,k)$ and a string $w \in \{0,1\}^{\ell(I,k)}$ and outputs a yes/no answer. We require that $(I,k) \in L$ if and only if there exists w for which $\mathcal{B}$ answers "yes".

For a function $f\colon \mathbb{N}^2 \to \mathbb{N}$ we say that the protocol has complexity f if (a) $\ell(I,k) \leq f(|I|,k)$ and (b) the running time of $\mathcal{B}$ on $s(I,k)+w$ is bounded by $f(|I|,k)$.

The following characterization was stated implicitly in [45, §1]. We provide a proof in the full version of the article for completeness.

▶ **Theorem 9.** *A parameterized problem L belongs to the class WK[1] if and only if:*
1. *$(I,k) \in L$ can be decided in time $2^{\mathsf{poly}(k)} \cdot |I|^{\mathcal{O}(1)}$, and*
2. *L admits a witness verification protocol of complexity $\mathsf{poly}(k, \log|I|)$.*

3 WK[1]-completeness of ILP

We summon several facts about the structure of solutions to ILP in the equality form. First we bound the support of a solution using the integral Carathéodory's Theorem.

▶ **Proposition 10** ([31, Corollary 5])**.** *Consider an optimization ILP of the form $\min\{c^T x\colon Ax = b, x \in \mathbb{Z}^n_{\geq 0}\}$ where $A \in \mathbb{Z}^{m\times n}, b \in \mathbb{Z}^m, c \in \mathbb{Z}^n$. If this program has a finite optimum, then it has an optimal solution x^* in which the number of non-zero entries is $\mathcal{O}(m(\log m + \log(\Delta))$ where Δ is the maximum absolute value of an entry in A and c.*

Next, we bound the ℓ_1-norm of a solution.

▶ **Proposition 11** ([51, Lemma 3])**.** *If the system $(Ax = b,\ x \in \mathbb{Z}^n_{\geq 0})$ is feasible, it has a solution x^* satisfying $||x^*||_1 \leq \mathcal{O}(m \cdot \Delta(A))^m \cdot (||b||_\infty + 1)$.*

At the expense of solving an LP relaxation, we can assume that the entries in the vector b are bounded. The following proposition is a corollary from the proximity bound by Eisenbrand and Weismantel [32].

▶ **Proposition 12** ([51, §4.2])**.** *There is a polynomial-time algorithm that, given $A \in \mathbb{Z}^{m\times n}$ and $b \in \mathbb{Z}^m$, returns a vector $b' \in \mathbb{Z}^m$ such that:*
1. *$||b'||_\infty = \mathcal{O}(m \cdot \Delta(A))^{m+1}$,*
2. *$(Ax = b,\ x \in \mathbb{Z}^n_{\geq 0})$ is feasible if and only if $(Ax = b',\ x \in \mathbb{Z}^n_{\geq 0})$ is feasible.*

We are ready to pin down the complexity status of ILP Feasibility.

▶ **Theorem 1.** Integer Linear Programming Feasibility *is* WK[1]*-complete when:*
- *parameterized by $m + \log(\Delta(A))$,*
- *parameterized by m only, under the assumption that $\Delta(A) = 1$ and the vector b is given in unary.*

Proof. We first show that the latter variant is WK[1]-hard. It admits a trivial PPT to the first variant, so then it remains to show that the first one belongs to WK[1].

We prove WK[1]-hardness by reducing from Set Cover parameterized by the universe size. This problem is known to be WK[1]-complete [45]. For each set $F \in \mathcal{F}$ we create a variable $x_F \in \mathbb{Z}_{\geq 0}$. For each $u \in U$ we write a constraint $\sum_{F \ni u} x_F \geq 1$, enforcing that u

must be covered. Then we add a constraint $\sum_{F \in \mathcal{F}} x_F \leq \ell$. It is easy to see that any integral solution corresponds to a vertex cover of size at most ℓ. This yields a polynomial parameter transformation: the number of constraints is $|U| + 1$, $\Delta(A) = 1$, and $||b||_\infty \leq |U|$.

Now we show that the first problem belongs to WK[1]. First we rewrite an ILP into the equality form (Observation 4). By Theorem 9 we have to check that the problem is solvable in time $2^{\mathsf{poly}(k)} \cdot s^{\mathcal{O}(1)}$ and that it admits a witness verification protocol with complexity $\mathsf{poly}(k, \log s)$, where s stands for the encoding size of the instance. The running time from Proposition 5 can be estimated as $2^{\mathcal{O}(k \log k)} \cdot s^{\mathcal{O}(1)}$ where $k = m + \Delta(A)$ so it suffices to design a witness verification protocol $(\mathcal{A}, \mathcal{B})$. The algorithm $\mathcal{A}$ preprocesses the ILP using Proposition 12 to ensure that $||b||_\infty = \mathcal{O}(m \cdot \Delta(A))^{m+1}$.

We apply Proposition 10 for the objective vector $c = 0$ so Δ equals $\Delta(A)$. We infer that if the ILP is feasible, then it admits a solution x^* with at most $\mathcal{O}(k^2)$ non-zero variables. By applying Proposition 12 to the ILP restricted to only these variables we can further assume that $||x^*||_1 \leq \mathcal{O}(m \cdot \Delta(A))^{2m+1} = 2^{\mathcal{O}(k^2 \log k)}$. Hence x^* can be encoded using $\mathcal{O}(k^2 \log s)$ bits for the choice of the non-zero variables and $\mathcal{O}(k^4 \log k)$ bits for their values. This gives an encoding of the witness supplied to the algorithm $\mathcal{B}$. Checking each constraint can be done in time polynomial with respect to the number of non-zero variables and the number of relevant bits, which is $\mathsf{poly}(k, \log s)$. This concludes the description of the witness verification protocol. ◀

By inspecting the reduction from Set Cover to ILP Feasibility, one can easily see that it also works with additional constraints $x_u \in \{0, 1\}$ for every $u \in U$. It follows that the variant with bounded variables is also WK[1]-hard, however its containment in WK[1] is not known and we find it unlikely (see Section 6).

▶ **Observation 13.** Binary ILP Feasibility *is* WK[1]*-hard when parameterized by* m*, under the assumption that* $\Delta(A) = 1$ *and the vector* b *is given in unary.*

4 Fast Witness Verification

We show containment in WK[1] for the problems listed in Theorem 2 using the criterion from Theorem 9. The first condition will be always satisfied by summoning a known FPT algorithm and we will focus on designing witness verification protocols.

4.1 Cut problems

In this section we treat r-Way Cut and Vertex Multiway Cut. We will employ connectivity oracles to quickly verify that a solution candidate forms a cut.

▶ **Definition 14.** *A* vertex (resp. edge) failure connectivity oracle *for a graph* G *and integer* d *supports two types of operations. An* update *operation is performed only once and it processes a set* D *of vertices (resp. edges) from* G *of size at most* d*. In a* query *operation, for given vertices* $u, v \in G$ *the oracle determines whether* u, v *are connected in* $G - D$*.*

▶ **Proposition 15** ([61])**.** *There is a deterministic vertex (resp. edge) failure connectivity oracle for undirected graphs with polynomial preprocessing time, update time* $\mathcal{O}(d^2 \cdot \log^4 n \cdot \log^4 d)$*, and worst-case query time* $\mathcal{O}(d)$*.*

The statement in [61] concerns vertex failure connectivity only but this setting generalizes edge failure connectivity. To see this, subdivide each edge in G and mimic removal of an edge by removing the corresponding vertex (see also [46]).

r-Way Cut	**Parameter:** k
Input: undirected graph G, integers r and k	
Question: is there a set $X \subseteq E(G)$ such that $G-X$ has at least r connected components and $\lvert X\rvert \leq k$?	

The vertex-deletion variant of the problem is W[1]-hard already under the parameterization by $k+r$ [63]. Edge deletion is easier and the above stated variant is FPT [19]. However, it does not admit a polynomial kernel unless NP $\subseteq$ coNP/poly [20]. We will show that r-Way Cut belongs to WK[1] which, in the light of Theorem 1, makes the problem amenable to ILP-based preprocessing.

▶ **Lemma 16.** r-Way Cut $\in$ WK[1].

Proof. By Theorem 9 we need to check that the problem is solvable in time $2^{\mathsf{poly}(k)}n^{\mathcal{O}(1)}$ and that it admits a witness verification protocol with complexity $\mathsf{poly}(k, \log n)$. The first condition is satisfied due to known $k^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$-time algorithm [19].

Observe that removal of one edge can increase the number of connected component by at most one. Let $\mathcal{C}(G)$ denote the set of connected components of G and $\ell = |\mathcal{C}(G)|$. If $\ell + k < r$ then clearly there can be no solution, so let us assume $r - \ell \leq k$. Suppose that $X \subseteq E(G)$ is a solution. For every $C \in \mathcal{C}(G)$ that is being split into $r_C > 1$ parts, there exists vertices $v_1, v_2, \ldots, v_{r_C} \in V(C)$ that are pairwise disconnected in $G - X$. The number of connected component of $G - X$ equals $\sum_{C \in \mathcal{C}(G)} r_C$. Since it suffices to split $r - \ell \leq k$ components, we may assume that there at most k components $C \in \mathcal{C}(G)$ for which $r_C > 1$. Therefore, when X is a solution, there must exist $\leq 2k$ vertices that can be divided into $t \leq k$ groups $V_1, V_2, \ldots, V_t$ so that (i) each group V_i is pairwise connected in G, (ii) $v \in V_i$ and $u \in V_j$ are not connected in G whenever $i \neq j$, (iii) distinct $u, v \in V_i$ are not connected in $G - X$, and (iv) $\sum_{i=1}^{t}(|V_i| - 1) = r - \ell$. This constitutes a certificate that X is a solution.

We now design the witness verification protocol $(\mathcal{A}, \mathcal{B})$. The algorithm $\mathcal{A}$ computes the connected components of G, assigns vertices in each component a unique label, and initializes the data structure from Proposition 15. The algorithm $\mathcal{B}$ inherits the state of the data structure together with labels marking the connected components and is provided the following witness: a set X of at most k edges and $t \leq k$ vertex sets $V_1, V_2, \ldots, V_t$ with $\leq 2k$ vertices in total. First we check that all the given vertices are pairwise distinct, in time $\mathcal{O}(k^2)$. Then we check the conditions (i)-(iv). The conditions (i)-(ii) can be verified using labels in time $\mathcal{O}(k^2)$. To handle condition (iii), we first perform the update operation on the edge failure connectivity oracle for set X in time $\mathcal{O}(k^3 \cdot \log^4 n)$. Then we query the oracle for each pair $u, v \in V_i$ for each $i \in [1, t]$ to check that they are disconnected in $G - X$. This takes total time $\mathcal{O}(k^3)$. Finally, condition (iv) boils down to number additions, which are performed in time $\mathcal{O}(k)$. This concludes the description of the protocol. ◀

We move on to another cut-based problem.

Vertex Multiway Cut	**Parameter:** k
Input: undirected graph G, set $T \subseteq V(G)$, integer k	
Question: is there a set $X \subseteq V(G) \setminus T$ such that each connected component of $G - X$ has at most one vertex from T and $\lvert X\rvert \leq k$?	

The existence of a polynomial kernel for Vertex Multiway Cut remains a major open problem [60]. Positive results include special cases where $|T| = \mathcal{O}(1)$ [60] or where the input graph is planar [49], and a quasi-polynomial kernel for the edge-deletion variant [70]. In addition, there exist reduction rules that compress the terminal set.

▶ **Proposition 17** ([21, Lemma 2.7]). *There is a polynomial-time algorithm that, given an instance* (G, T, k) *of* Vertex Multiway Cut*, outputs an equivalent instance* (G', T', k') *with* $k' \leq k$ *and* $|T'| \leq 2k'$.

Our witness verification protocol combines this procedure with the vertex failure connectivity oracle.

▶ **Lemma 18.** Vertex Multiway Cut $\in$ WK[1].

Proof. Vertex Multiway Cut can be solved in time $2^k \cdot n^{\mathcal{O}(1)}$ [21] and it remains to construct a witness verification protocol. The algorithm $\mathcal{A}$ executes the reduction from Proposition 17 and so we can assume $|T| \leq 2k$. Then it initializes the vertex failure connectivity oracle from Proposition 15. The algorithm $\mathcal{B}$ is given the reduced instance, the state of the oracle, and the witness as simply a solution candidate $X \subseteq V(G)$, $|X| \leq k$. First it checks if $X \cap T = \emptyset$ in time $\mathcal{O}(k^2)$. Then it performs the update operation on the oracle for set X, in time $\mathcal{O}(k^3 \cdot \log^4 n)$. Finally, it asks oracle for each pair $u, v \in T$ whether u, v are disconnected in $G - X$, in total time $\mathcal{O}(k^3)$. Hence the complexity of the protocol is $\mathsf{poly}(k, \log n)$. ◀

4.2 Minimum Common String Partition

For two strings x, y of equal length, their *common string partition* of size k is given by partitions $x = x_1 x_2 \dots x_k$, $y = y_1 y_2 \dots y_k$, such that there exists a permutation $f \colon [k] \to [k]$ satisfying $x_i = y_{f(i)}$ for each $i \in [1, k]$.

Minimum Common String Partition	**Parameter:** k
Input: strings x, y of equal length, integer k	
Question: is there a common string partition of x, y of size at most k?	

The study of this problem is motivated by applications in comparative genomics [12]. To the best of our knowledge, nothing is known about its kernelization status. In order to design a witness verification protocol, we will employ the following data structure for maintaining a dynamic collection of strings.

▶ **Proposition 19** ([64]). *There is a deterministic data structure that supports the following operations in worst-case time* $\mathcal{O}(\log^2 m \cdot \log^* m)$ *where* m *is the total number of operations.*

1. *Create a singleton string with symbol* z.
2. *Create a new string by concatenating two strings from the collection.*
3. *Create two new strings by splitting a string from the collection at the given index.*
4. *Check if two strings in the collection are equal.*

Moreover, operations $(2, 3)$ *keep the host strings in the collection.*

▶ **Lemma 20.** Minimum Common String Partition $\in$ WK[1].

Proof. The problem can be solved in time $2^{\mathcal{O}(k^2 \log k)} \cdot n^{\mathcal{O}(1)}$ [12] and so it suffices to give a witness verification protocol of complexity $\mathsf{poly}(k, \log n)$. The algorithm $\mathcal{A}$ builds strings x, y using operations $(1, 3)$ of data structure from Proposition 19, then it outputs the computed data structure.

A witness is a partition $x = x_1 x_2 \dots x_k$, specified by k indices from the range $[1, n]$, together with a permutation $f \colon [k] \to [k]$. The algorithm $\mathcal{B}$ inherits the state of the data structure from $\mathcal{A}$ and creates strings $x_1, x_2, \dots, x_k$ using the split operation. Then it concatenates them in the order given by f and checks if the resulting string is equal to y.

The witness can be encoded using $k \log n + k \log k$ bits. The verification process requires $2k + 1$ operations on the data structure, each taking time $\mathcal{O}(\log^2 n \cdot \log^* n)$. ◀

4.3 Long Path

A *feedback vertex set* (FVS) in a graph G is a set $X \subseteq V(G)$ for which $G - X$ is acyclic. The FVS number of G is the size of its smallest feedback vertex set.

Long Path / FVS	**Parameter:** k = FVS number of G
Input: undirected graph G, integer ℓ	
Question: is there a path on ℓ vertices in G?	

It is known that Long Path admits a polynomial kernel when parameterized by the vertex cover number [8] and in the same paper it was asked whether the same holds for the FVS number parameterization. To the best of our knowledge, this question remains open since 2013. An FPT algorithm for Long Path / FVS is implied by the following result because treewidth is never larger that the FVS number.

▶ **Proposition 21** ([5, Theorem 3.20]). Long Path *can be solved in time* $2^{\mathcal{O}(\mathsf{tw})} \cdot n^{\mathcal{O}(1)}$ *where* tw *denotes the treewidth of the input graph.*

We remark that this result has been stated under the assumption that a tree decomposition of width tw is provided in the input but this assumption can be dropped due to known approximation algorithms for treewidth [5, §5].

For a tree T and $u, v \in V(T)$ let $P_T(u, v)$ denote the unique (u, v)-path in T. The length of a path P is defined as $|V(P)|$.

▶ **Lemma 22.** *The problem* Long Path / FVS *belongs to* WK[1].

Proof. The FPT algorithm follows from Proposition 21 since $\mathsf{tw}(G) \leq \mathsf{fvs}(G)$. We construct a witness verification protocol $(\mathcal{A}, \mathcal{B})$. The algorithm $\mathcal{A}$ first runs 2-approximation for Feedback Vertex Set [2] so we can assume that a feedback vertex set X of size at most $2k$ is known. Next, it computes the connected components of $G - X$ by assigning vertices in each component a unique label. Then, on each tree T we compute the following information. For each $v_1, v_2 \in V(T)$ we store the length of the path $P_T(v_1, v_2)$. For each $v_1, v_2, u_1, u_2 \in V(T)$ we store a bit indicating whether $V(P_T(v_1, v_2)) \cap V(P_T(u_1, u_2)) = \emptyset$. The algorithm $\mathcal{A}$ outputs all these structures, together with the adjacency matrix of G.

Observe that when P is a path in G, then it has at most $2k + 1$ maximal subpaths $P_1, P_2, \ldots, P_t$ in $G - X$. The witness is a description of a path P as a sequence of vertices from X and pairs of vertices from $G - X$ that form the endpoints of subpaths P_i. To verify the witness we need to check (i) that the total length of path is ℓ, (ii) that the consecutive vertices in the description are adjacent in G, and (iii) that the given subpaths are pairwise disjoint. To check condition (i) we read the length of each path P_i specified by endpoints v_1, v_2. This quantity can be retrieved from the precomputed table and the total length is then computed using $\mathcal{O}(k)$ addition. Condition (ii) is tested with the adjacency matrix. To check condition (iii), we consider all pairs $i, j \in [1, t]$ and if the endpoints of P_i, P_j belong to the same tree, we determine disjointedness of P_i, P_j using the precomputed table. Then we also compare the listed vertices from X to ensure that they are pairwise distinct. The witness can be encoded with $\mathcal{O}(k \log n)$ bits while the verification takes time $\mathcal{O}(k^2)$. This yields a witness verification protocol of complexity $\mathsf{poly}(k, \log n)$. ◀

4.4 Steiner Tree

Steiner Tree / T	**Parameter:** $k = \lvert T \rvert$
Input: graph G, set $T \subseteq V(G)$, integer ℓ	
Question: is there a tree $X \subseteq E(G)$ that spans T and $\lvert X \rvert \leq \ell$?	

Hermelin et al. [45] showed that Steiner Tree is WK[1]-complete when parameterized by ℓ. Since $k \leq \ell$ in any non-trivial instance, this implies that the problem is WK[1]-hard when parameterized by k and, furthermore, that it does not admit a polynomial kernel unless NP $\subseteq$ coNP/poly.

▶ **Lemma 23.** *The problem* Steiner Tree / T *belongs to WK[1].*

Proof. Steiner Tree can be solved in time $3^k \cdot n^{\mathcal{O}(1)}$ [26] and we present a witness verification protocol. The algorithm $\mathcal{A}$ computes all the pairwise distances in G and outputs them together with the set T. The witness supplied to algorithm $\mathcal{B}$ is given by a set $Y \subseteq V(G) \setminus T$ of at most $k-1$ vertices and a tree F on the vertex set $T \cup Y$. The verification protocol reads the distances in G between each pair $u, v \in T \cup Y$ where $uv \in E(F)$ and checks if their sum is at most ℓ.

If the verification succeeds then clearly G contains a connected subgraph spanning T on $\leq \ell$ edges, so it also contains a tree with this property. In the other direction, suppose that there exists a solution tree X. This tree has at most k leaves so the number of vertices of degree greater than 2 is at most $k-1$. The witness guesses such vertices that are not in T and the tree F obtained from X by contracting each vertex $V(X) \setminus T$ of degree two to one of its neighbors. The total number of edges lying on paths corresponding to $E(F)$ is clearly $|X|$ which bounds the sum of distances read by the algorithm $\mathcal{B}$. This concludes the correctness proof of the protocol. The witness can be encoded using $\mathcal{O}(k \log n)$ bits while verification requires $\mathcal{O}(k)$ additions of numbers from $[1, n]$. ◀

4.5 Optimal Discretization

Consider sets $W_1, W_2 \subseteq \mathbb{Q}^2$. A pair (X, Y) of sets $X, Y \subseteq \mathbb{Q}$ is called a separation of (W_1, W_2) if for every $(x_1, y_1) \in W_1$ and $(x_2, y_2) \in W_2$ there exists an element of X strictly between x_1 and x_2 or an element of Y strictly between y_1 and y_2. A geometric interpretation of (X, Y) is a family of axis-parallel lines (vertical line ℓ_x for each $x \in X$ and horizontal line ℓ_y for each $y \in Y$) so that for every component C of $\mathbb{R}^2 \setminus \bigcup_{z \in X \cup Y} \ell_z$ the closure of C contains either only points from W_1 or only points from W_2.

Optimal Discretization	**Parameter:** k
Input: sets $W_1, W_2 \subseteq \mathbb{Q}^2$, integer k	
Question: is there a separation (X, Y) of (W_1, W_2) with $\lvert X\rvert + \lvert Y\rvert \leq k$?	

The geometric interpretation makes the problem attractive from the perspective of learning theory. This motivated the study of parameterized algorithms for Optimal Discretization: it turns out to be FPT [58] but only when we consider axis-parallel separating lines [10]. To the best of our knowledge, nothing is known about its kernelization status.

For $F \subseteq \mathbb{Q}$ we define $h(W) = \{\frac{x_1+x_2}{2} \colon x_1, x_2 \in F\}$. For $W \subseteq \mathbb{Q}^2$ we define sets $h_X(W), h_Y(W) \subseteq \mathbb{Q}$ as respectively $h(W_X), g(W_Y)$, where W_X, W_Y are projections of W into X/Y-coordinates. It is easy to modify any separation (X, Y) of (W_1, W_2) to only use coordinates that lie exactly in between two coordinates from the input, without affecting the set of pairs that are separated.

▶ **Observation 24.** *There exists a minimum-size separation (X, Y) of (W_1, W_2) satisfying $X \subseteq h_X(W_1 \cup W_2)$, $Y \subseteq h_Y(W_1 \cup W_2)$.*

▶ **Lemma 25.** Optimal Discretization $\in$ WK[1].

Proof. First we check that Optimal Discretization can be solved in time $2^{\mathcal{O}(k^2 \log k)} n^{\mathcal{O}(1)}$ [58]. The algorithm $\mathcal{A}$ of the witness verification protocol considers all tuples (x_1, x_2, y_1, y_2) where $x_1, x_2 \in h_X(W_1 \cup W_2) \cup \{-\infty, \infty\}$, $y_1, y_2 \in h_Y(W_1 \cup W_2) \cup \{-\infty, \infty\}$, $x_1 < x_2$, and $y_1 < y_2$. Their number is polynomial in $|W_1| + |W_2|$. For a tuple (x_1, x_2, y_1, y_2) it checks whether $[x_1, x_2] \times [y_1, y_2]$ contains at least one point from W_1 and from W_2. If it contains both, then such tuple is marked as bad. The algorithm $\mathcal{A}$ outputs the set of bad tuples, sorted lexicographically.

The witness supplied to the algorithm $\mathcal{B}$ is a pair (X, Y) where $X \subseteq h_X(W_1 \cup W_2)$, $Y \subseteq h_Y(W_1 \cup W_2)$, and $|X| + |Y| \leq k$. By Observation 24 we can assume that there exists an optimal solution of this form. Moreover, we require the sets X, Y to be listed in the increasing order of elements. The verification mechanism considers all tuples (x_1, x_2, y_1, y_2) where x_1, x_2 are consecutive elements in $X \cup \{-\infty, \infty\}$ and likewise for y_1, y_2. There are $\mathcal{O}(k^2)$ such tuples, each corresponding to a connected component of $\mathbb{R}^2 \setminus \bigcup_{z \in X \cup Y} \ell_z$. Then (X, Y) forms a separation for (W_1, W_2) if and only if none of these tuples has been marked as bad. One can check if (x_1, x_2, y_1, y_2) is bad by performing binary search in the data structure provided by the algorithm $\mathcal{A}$. In summary, the witness can be encoded with $\mathcal{O}(k \cdot \log n)$ bits whereas verification takes time $\mathcal{O}(k^2 \cdot \log n)$. ◀

5 Weighted Vertex Cover

The *vertex cover number* (VC number) of G is the size of the smallest vertex cover in G.

Weighted Vertex Cover / VC	**Parameter:** VC number of G
Input: graph G, weight function $w\colon V(G) \to \mathbb{Z}_{\geq 0}$ given in unary, integer ℓ	
Question: is there a vertex cover in G of total weight at most ℓ?	

The unweighted variant of Vertex Cover is probably the most heavily studied problem in kernelization [16, 35, 38, 57]. The weighted variant has a polynomial kernel when parameterized by the number of vertices in the sought solution [34] but it is unlikely to admit a polynomial kernel when parameterized by the VC number, even when the weights are given in unary [47]. We remark that if the weights were provided in binary, the theorem below would also hold but only when treating the logarithm of the maximal weight as an additional parameter.

▶ **Theorem 3.** Weighted Vertex Cover / VC *admits a linear parameter transformation to* Binary ILP Feasibility *parameterized by* $m + \log(\Delta(A))$. *Furthermore, the problem admits a linear parameter transformation to* MILP Feasibility *parametrized by the number of integral variables.*

Proof. We can assume that a vertex cover Y of G of size at most $2k$ is known, due to the classic 2-approximation algorithm. The problem can be readily solved in time $2^{|Y|} \cdot |V(G)|$ so by the standard argument we can assume that $\log n \leq k$, where n is the total input size. Consider the following mixed integer linear program.

$$\begin{aligned}
&\min \sum_{v \in V(G)} w(v) \cdot x_v \\
\forall_{u \in Y} \quad &\sum_{v \in N_G(u)} x_v + \deg_G(u) \cdot x_u \geq \deg_G(u) \\
\forall_{v \in V(G)} \quad &0 \leq x_v \leq 1 \\
\forall_{u \in Y} \quad &x_u \in \mathbb{Z}
\end{aligned}$$

The number of integral variables, as well as the number of constrains, is $|Y| \leq 2k$. Recall that the bounds $0 \leq x_v \leq 1$ are not counted as constraints in our setup.

We claim that the optimum of this program equals the minimum weight vertex cover in G. First, suppose that $S \subseteq V(G)$ is a vertex cover. We claim that the binary characteristic vector x of S satisfies all the constraints. Consider $u \in Y$. If $u \in S$ then clearly $\sum_{v \in N_G(u)} x_v + \deg_G(u) \cdot x_u \geq \deg_G(u)$. If $u \notin S$ then $N_G(u) \subseteq S$ and again the inequality holds.

In the other direction, consider a feasible solution x. We claim that set $S = \{v \in V(G) \colon x_v = 1\}$ forms a vertex cover. We need to show that every edge $uw \in E(G)$ is covered by S. Since Y is also a vertex cover, we can assume w.l.o.g. that $u \in Y$. Suppose that $x_u < 1$. Because x_u is an integral variable, this implies $x_u = 0$, and so it must hold that $\sum_{v \in N_G(u)} x_v \geq \deg_G(u)$. All the variables are upper bounded by 1, which enforces that $x_v = 1$ for every $v \in N_G(u)$, in particular for $v = w$. Hence the presented MILP indeed models minimum weight vertex cover.

In order to obtain a feasibility problem we replace the objective function with a constraint $\sum_{v \in V(G)} w(v) \cdot x_v \leq \ell$. This proves the second part of the theorem. Observe that the entries in the obtained ILP matrix are vertex degrees and vertex weights, which are at most 2^k because we assumed $\log n \leq k$. This gives the bound on $\Delta(A)$ and proves the first part of the theorem, by simply treating all the variables as integral. ◀

It is tempting to try to get rid of the upper bounds $x_v \leq 1$ and attain an ILP in the form considered in Theorem 1. If we allow for not $\mathcal{O}(k)$ but $\mathsf{poly}(k)$ constraints, this is equivalent to proving that Weighted Vertex Cover / VC is in WK[1]. A potential witness verification protocol could guess the intersection of a solution S with the vertex cover Y but it is unclear how to verify quickly that the remaining part of the solution has bounded weight.

6 Conclusion

We have presented a characterization of parameterized problems reducible to ILP with few constraints and classified several new problems into this category through witness verification protocols. We find this connection surprising and inspiring for exploring new directions in parameterized complexity. Can we further extend this classification? In particular, we ask whether the following problems belong to WK[1]: Weighted Vertex Cover / VC, Connected FVS, Directed FVS, and Vertex Planarization (the last three parameterized by the solution size).

Our classification concerns ILPs in the standard form without upper bounds. Allowing for upper bounds on variables gives a greater expressive power when the number of constraints is a parameter (see Observation 13 and also [68]). It would be interesting to also characterize parameterized problems reducible to upper-bounded ILP with few constraints. However, the location of such a feasibility problem in the parameterized complexity landscape remains unclear because we do not know whether it admits short NP-witnesses [72]. Yet another direction is to describe problems that are reducible to ILP with few variables.

Containment in WK[1] only ensures that a problem can be modeled by ILP with $\mathsf{poly}(k)$ constraints. Even though the degree of this polynomial can be retraced from the chain of reductions, it is unlikely to be optimal. As the natural next step towards bridging the gap between theory and practice, we would like to optimize the polynomial degree, by designing explicit ILP formulation with $\mathcal{O}(k^c)$ constraints, where c is as small as possible, ideally one. A good starting point would be to consider problems with simple proofs of containment in WK[1], such as k-Path or Connected Vertex Cover [45].

References

1 Stephan Artmann, Robert Weismantel, and Rico Zenklusen. A strongly polynomial algorithm for bimodular integer linear programming. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, pages 1206–1219, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3055399.3055473.

2 Vineet Bafna, Piotr Berman, and Toshihiro Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM Journal on Discrete Mathematics*, 12(3):289–297, 1999. doi:10.1137/S0895480196305124.

3 Max Bannach, Florian Chudigiewitsch, Kim-Manuel Klein, and Marcel Wienöbst. PACE Solver Description: UzL Exact Solver for One-Sided Crossing Minimization. In Édouard Bonnet and Paweł Rzążewski, editors, *19th International Symposium on Parameterized and Exact Computation (IPEC 2024)*, volume 321 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 28:1–28:4, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.IPEC.2024.28.

4 Moritz Bergenthal, Jona Dirks, Thorben Freese, Jakob Gahde, Enna Gerhard, Mario Grobler, and Sebastian Siebertz. PACE Solver Description: GraPA-JAVA. In Holger Dell and Jesper Nederlof, editors, *17th International Symposium on Parameterized and Exact Computation (IPEC 2022)*, volume 249 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:4, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.IPEC.2022.30.

5 Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Information and Computation*, 243:86–111, 2015. 40th International Colloquium on Automata, Languages and Programming (ICALP 2013). doi:10.1016/j.ic.2014.12.008.

6 Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *Journal of Computer and System Sciences*, 75(8):423–434, 2009. doi:10.1016/j.jcss.2009.04.001.

7 Hans L. Bodlaender, Carla Groenland, Jesper Nederlof, and Céline M. F. Swennenhuis. Parameterized problems complete for nondeterministic FPT time and logarithmic space. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 193–204. IEEE, 2021. doi:10.1109/FOCS52979.2021.00027.

8 Hans L Bodlaender, Bart MP 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.

9 Kimon Boehmer, Lukas Lee George, Fanny Hauser, and Jesse Palarus. PACE Solver Description: Arcee. In *19th International Symposium on Parameterized and Exact Computation (IPEC 2024)*, volume 321 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 33:1–33:4, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.IPEC.2024.33.

10 Édouard Bonnet, Panos Giannopoulos, and Michael Lampis. On the Parameterized Complexity of Red-Blue Points Separation. In Daniel Lokshtanov and Naomi Nishimura, editors, *12th International Symposium on Parameterized and Exact Computation (IPEC 2017)*, volume 89 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.IPEC.2017.8.

11 Cornelius Brand, Martin Koutecký, and Sebastian Ordyniak. Parameterized algorithms for MILPs with small treedepth. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 12249–12257. AAAI Press, 2021. doi:10.1609/AAAI.V35I14.17454.

12 Laurent Bulteau and Christian Komusiewicz. Minimum common string partition parameterized by partition size is fixed-parameter tractable. In Chandra Chekuri, editor, *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 102–121. SIAM, 2014. doi:10.1137/1.9781611973402.8.

13 Kevin Burrage, Vladimir Estivill-Castro, Michael Fellows, Michael Langston, Shev Mac, and Frances Rosamond. The undirected feedback vertex set problem has a poly(k) kernel. In Hans L. Bodlaender and Michael A. Langston, editors, *Parameterized and Exact Computation*, pages 192–202, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. doi:10.1007/11847250_18.

14 Der-San Chen, Robert G Batson, and Yu Dang. *Applied Integer Programming: Modeling and Solution*. Wiley Online Library, 2009. doi:10.1002/9781118166000.

15 Hua Chen, Lin Chen, and Guochuan Zhang. FPT algorithms for a special block-structured integer program with applications in scheduling. *Mathematical Programming*, 208(1):463–496, 2024. doi:10.1007/s10107-023-02046-z.

16 Benny Chor, Mike Fellows, and David Juedes. Linear kernels in linear time, or how to save k colors in $O(n^2)$ steps. In *Graph-Theoretic Concepts in Computer Science: 30th International Workshop, WG 2004, Bad Honnef, Germany, June 21-23, 2004. Revised Papers 30*, pages 257–269. Springer, 2005. doi:10.1007/978-3-540-30559-0_22.

17 Jana Cslovjecsek, Martin Koutecký, Alexandra Lassota, Michał Pilipczuk, and Adam Polak. Parameterized algorithms for block-structured integer programs with large entries. 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 740–751. SIAM, 2024. doi:10.1137/1.9781611977912.29.

18 William H. Cunningham and Jim Geelen. On integer programming and the branch-width of the constraint matrix. In Matteo Fischetti and David P. Williamson, editors, *Integer Programming and Combinatorial Optimization*, pages 158–166, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. doi:10.1007/978-3-540-72792-7_13.

19 Marek Cygan, Paweł Komosa, Daniel Lokshtanov, Marcin Pilipczuk, Michał Pilipczuk, Saket Saurabh, and Magnus Wahlström. Randomized contractions meet lean decompositions. *ACM Transactions on Algorithms (TALG)*, 17(1):1–30, 2020. doi:10.1145/3426738.

20 Marek Cygan, Stefan Kratsch, Marcin Pilipczuk, Michał Pilipczuk, and Magnus Wahlström. Clique cover and graph separation: New incompressibility results. *ACM Trans. Comput. Theory*, 6(2), May 2014. doi:10.1145/2594439.

21 Marek Cygan, Marcin Pilipczuk, Michal Pilipczuk, and Jakub Onufry Wojtaszczyk. On multiway cut parameterized above lower bounds. *ACM Trans. Comput. Theory*, 5(1):3:1–3:11, 2013. doi:10.1145/2462896.2462899.

22 George Bernard Dantzig, Delbert R Fulkerson, and Selmer Martin Johnson. On a linear-programming, combinatorial approach to the traveling-salesman problem. *Operations Research*, 7(1):58–66, 1959. doi:10.1287/opre.7.1.58.

23 Jona Dirks, Mario Grobler, Roman Rabinovich, Yannik Schnaubelt, Sebastian Siebertz, and Maximilian Sonneborn. PACE solver description: PACA-JAVA. In Petr A. Golovach and Meirav Zehavi, editors, *16th International Symposium on Parameterized and Exact Computation, IPEC 2021, September 8-10, 2021, Lisbon, Portugal*, volume 214 of *LIPIcs*, pages 30:1–30:4. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.IPEC.2021.30.

24 Alexander Dobler. PACE Solver Description: CRGone. In Édouard Bonnet and Paweł Rzążewski, editors, *19th International Symposium on Parameterized and Exact Computation (IPEC 2024)*, volume 321 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 29:1–29:4, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.IPEC.2024.29.

25 Rodney G. Downey and Michael R. Fellows. Parameterized computational feasibility. In Peter Clote and Jeffrey B. Remmel, editors, *Feasible Mathematics II*, pages 219–244, Boston, MA, 1995. Birkhäuser Boston. doi:10.1007/978-1-4612-2566-9_7.

26 S. E. Dreyfus and R. A. Wagner. The steiner problem in graphs. *Networks*, 1(3):195–207, 1971. doi:10.1002/net.3230010302.

27 Andrew Drucker, Jesper Nederlof, and Rahul Santhanam. Exponential Time Paradigms Through the Polynomial Time Lens. In Piotr Sankowski and Christos Zaroliagis, editors, *24th Annual European Symposium on Algorithms (ESA 2016)*, volume 57 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 36:1–36:14, Dagstuhl, Germany, 2016. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2016.36.

28 Pavel Dvořák, Eduard Eiben, Robert Ganian, Dušan Knop, and Sebastian Ordyniak. Solving integer linear programs with a small number of global variables and constraints. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, IJCAI'17, pages 607–613. AAAI Press, 2017. doi:10.5555/3171642.3171730.

29 M. Ayaz Dzulfikar, Johannes K. Fichte, and Markus Hecher. The PACE 2019 Parameterized Algorithms and Computational Experiments Challenge: The Fourth Iteration. In Bart M. P. Jansen and Jan Arne Telle, editors, *14th International Symposium on Parameterized and Exact Computation (IPEC 2019)*, volume 148 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:23, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.IPEC.2019.25.

30 Friedrich Eisenbrand, Christoph Hunkenschröder, Kim-Manuel Klein, Martin Koutecký, Asaf Levin, and Shmuel Onn. Sparse integer programming is fixed-parameter tractable. *Math. Oper. Res.*, 50(3):2141–2156, 2025. doi:10.1287/MOOR.2023.0162.

31 Friedrich Eisenbrand and Gennady Shmonin. Carathéodory bounds for integer cones. *Oper. Res. Lett.*, 34(5):564–568, 2006. doi:10.1016/J.ORL.2005.09.008.

32 Friedrich Eisenbrand and Robert Weismantel. Proximity results and faster algorithms for integer programming using the Steinitz lemma. *ACM Trans. Algorithms*, 16(1):5:1–5:14, 2020. doi:10.1145/3340322.

33 Michael Elberfeld, Christoph Stockhusen, and Till Tantau. On the space and circuit complexity of parameterized problems: Classes and completeness. *Algorithmica*, 71(3):661–701, 2015. doi:10.1007/S00453-014-9944-Y.

34 Michael Etscheid, Stefan Kratsch, Matthias Mnich, and Heiko Röglin. Polynomial kernels for weighted problems. *J. Comput. Syst. Sci.*, 84:1–10, 2017. doi:10.1016/j.jcss.2016.06.004.

35 Michael R. Fellows, Bart M.P. Jansen, and Frances Rosamond. Towards fully multivariate algorithmics: Parameter ecology and the deconstruction of computational complexity. *European Journal of Combinatorics*, 34(3):541–566, 2013. Combinatorial Algorithms and Complexity. doi:10.1016/j.ejc.2012.04.008.

36 Samuel Fiorini, Gwenaël Joret, Stefan Weltge, and Yelena Yuditsky. Integer programs with bounded subdeterminants and two nonzeros per row. *J. ACM*, 72(1), January 2025. doi:10.1145/3695985.

37 Christodoulos A Floudas and Xiaoxia Lin. Mixed integer linear programming in process scheduling: Modeling, algorithms, and applications. *Annals of Operations Research*, 139:131–162, 2005. doi:10.1007/s10479-005-3446-x.

38 Fedor Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: theory of parameterized preprocessing*. Cambridge University Press, 2019. doi:10.1017/9781107415157.

39 Fedor V Fomin, Fahad Panolan, Maadapuzhi Sridharan Ramanujan, and Saket Saurabh. On the optimality of pseudo-polynomial algorithms for integer programming. *Mathematical Programming*, 198(1):561–593, 2023. doi:10.1007/s10107-022-01783-x.

40 Robert Ganian and Sebastian Ordyniak. The complexity landscape of decompositional parameters for ilp. *Artificial Intelligence*, 257:61–71, 2018. doi:10.1016/j.artint.2017.12.006.

41 Tomáš Gavenčiak, Martin Koutecký, and Dušan Knop. Integer programming in parameterized complexity: Five miniatures. *Discrete Optimization*, 44:100596, 2022. Optimization and Discrete Geometry. doi:10.1016/j.disopt.2020.100596.

42 Michel X Goemans and Young-Soo Myung. A catalog of steiner tree formulations. *Networks*, 23(1):19–28, 1993. doi:10.1002/net.3230230104.

43 Ernestine Großmann, Tobias Heuer, Christian Schulz, and Darren Strash. The PACE 2022 Parameterized Algorithms and Computational Experiments Challenge: Directed Feedback Vertex Set. In Holger Dell and Jesper Nederlof, editors, *17th International Symposium on Parameterized and Exact Computation (IPEC 2022)*, volume 249 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:18, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.IPEC.2022.26.

44 Danny Hermelin, Shlomo Karhi, Michael Pinedo, and Dvir Shabtay. New algorithms for minimizing the weighted number of tardy jobs on a single machine. *Annals of Operations Research*, 298:271–287, 2021. doi:10.1007/s10479-018-2852-9.

45 Danny Hermelin, Stefan Kratsch, Karolina Soltys, Magnus Wahlström, and Xi Wu. A Completeness Theory for Polynomial (Turing) Kernelization. *Algorithmica*, 71(3):702–730, 2015. doi:10.1007/S00453-014-9910-8.

46 Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, July 2001. doi:10.1145/502090.502095.

47 Bart M. P. Jansen and Hans L. Bodlaender. Vertex cover kernelization revisited - upper and lower bounds for a refined parameter. *Theory Comput. Syst.*, 53(2):263–299, 2013. doi:10.1007/S00224-012-9393-4.

48 Bart M. P. Jansen and Stefan Kratsch. A structural approach to kernels for ilps: Treewidth and total unimodularity. In Nikhil Bansal and Irene Finocchi, editors, *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, volume 9294 of *Lecture Notes in Computer Science*, pages 779–791. Springer, 2015. doi:10.1007/978-3-662-48350-3_65.

49 Bart M. P. Jansen, Marcin Pilipczuk, and Erik Jan van Leeuwen. A deterministic polynomial kernel for odd cycle transversal and vertex multiway cut in planar graphs. *SIAM J. Discret. Math.*, 35(4):2387–2429, 2021. doi:10.1137/20M1353782.

50 Klaus Jansen, Alexandra Lassota, and Lars Rohwedder. Near-linear time algorithm for n-fold ILPs via color coding. *SIAM Journal on Discrete Mathematics*, 34(4):2282–2299, 2020. doi:10.1137/19M1303873.

51 Klaus Jansen and Lars Rohwedder. On Integer Programming and Convolution. In Avrim Blum, editor, *10th Innovations in Theoretical Computer Science Conference (ITCS 2019)*, volume 124 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 43:1–43:17, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ITCS.2019.43.

52 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.

53 Leon Kellerhals, Tomohiro Koana, André Nichterlein, and Philipp Zschoche. The PACE 2021 Parameterized Algorithms and Computational Experiments Challenge: Cluster Editing. In Petr A. Golovach and Meirav Zehavi, editors, *16th International Symposium on Parameterized and Exact Computation (IPEC 2021)*, volume 214 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:18, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.IPEC.2021.26.

54 Philipp Kindermann, Fabian Klute, and Soeren Terziadis. The PACE 2024 Parameterized Algorithms and Computational Experiments Challenge: One-Sided Crossing Minimization. In Édouard Bonnet and Paweł Rzążewski, editors, *19th International Symposium on Parameterized and Exact Computation (IPEC 2024)*, volume 321 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:20, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.IPEC.2024.26.

55 Dušan Knop and Martin Koutecký. Scheduling meets n-fold integer programming. *Journal of Scheduling*, 21:493–503, 2018. doi:10.1007/S10951-017-0550-0.

56 Stefan Kratsch. On polynomial kernels for integer linear programs: Covering, packing and feasibility. In Hans L. Bodlaender and Giuseppe F. Italiano, editors, *Algorithms – ESA 2013*, pages 647–658, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. doi:10.1007/978-3-642-40450-4_55.

57 Stefan Kratsch. A randomized polynomial kernelization for vertex cover with a smaller parameter. *SIAM Journal on Discrete Mathematics*, 32(3):1806–1839, 2018. doi:10.1137/16M1104585.

58 Stefan Kratsch, Tomás Masarík, Irene Muzi, Marcin Pilipczuk, and Manuel Sorge. Optimal discretization is fixed-parameter tractable. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 1702–1719. SIAM, 2021. doi:10.1137/1.9781611976465.103.

59 Stefan Kratsch and Vuong Anh Quyen. On kernels for covering and packing ilps with small coefficients. In Marek Cygan and Pinar Heggernes, editors, *Parameterized and Exact Computation*, pages 307–318, Cham, 2014. Springer International Publishing. doi:10.1007/978-3-319-13524-3_26.

60 Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. *J. ACM*, 67(3):16:1–16:50, 2020. doi:10.1145/3390887.

61 Yaowei Long and Thatchaphol Saranurak. Near-optimal deterministic vertex-failure connectivity oracles. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 1002–1010. IEEE, 2022. doi:10.1109/FOCS54457.2022.00098.

62 George S Lueker. *Two NP-complete problems in nonnegative integer programming*. Princeton University. Department of Electrical Engineering, 1975.

63 Dániel Marx. Parameterized graph separation problems. *Theoretical Computer Science*, 351(3):394–406, 2006. Parameterized and Exact Computation. doi:10.1016/j.tcs.2005.10.007.

64 Kurt Mehlhorn, R. Sundar, and Christian Uhrig. Maintaining dynamic sequences under equality tests in polylogarithmic time. *Algorithmica*, 17(2):183–198, 1997. doi:10.1007/BF02522825.

65 C. E. Miller, A. W. Tucker, and R. A. Zemlin. Integer programming formulation of traveling salesman problems. *J. ACM*, 7(4):326–329, October 1960. doi:10.1145/321043.321046.

66 George Nemhauser and Laurence Wolsey. *The Scope of Integer and Combinatorial Optimization*, chapter I.1, pages 1–26. John Wiley and Sons, Ltd, 1988. doi:10.1002/9781118627372.ch1.

67 Michał Pilipczuk and Marcin Wrochna. On space efficiency of algorithms working on structural decompositions of graphs. *ACM Trans. Comput. Theory*, 9(4):18:1–18:36, 2018. doi:10.1145/3154856.

68 Lars Rohwedder and Karol Węgrzycki. Fine-Grained Equivalence for Problems Related to Integer Linear Programming. In Raghu Meka, editor, *16th Innovations in Theoretical Computer Science Conference (ITCS 2025)*, volume 325 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 83:1–83:18, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ITCS.2025.83.

69 Stéphan Thomassé. A 4k2 kernel for feedback vertex set. *ACM Trans. Algorithms*, 6(2), April 2010. doi:10.1145/1721837.1721848.

70 Magnus Wahlström. Quasipolynomial multicut-mimicking networks and kernels for multiway cut problems. *ACM Trans. Algorithms*, 18(2):15:1–15:19, 2022. doi:10.1145/3501304.

71 Justin C Williams. A linear-size zero – One programming model for the minimum spanning tree problem in planar graphs. *Networks: An International Journal*, 39(1):53–60, 2002. doi:10.1002/net.10010.

72 Michal Wlodarczyk. Does subset sum admit short proofs? 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 58:1–58:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ISAAC.2024.58.

Treedepth Inapproximability and Exponential ETH Lower Bound

Édouard Bonnet
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Daniel Neuen
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Marek Sokołowski
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Abstract

Treedepth is a central parameter to algorithmic graph theory. The current state-of-the-art in computing and approximating treedepth consists of a $2^{O(k^2)}n$-time exact algorithm and a polynomial-time $O(\mathrm{OPT}\log^{3/2}\mathrm{OPT})$-approximation algorithm, where the former algorithm returns an *elimination forest* of height k (witnessing that treedepth is at most k) for the n-vertex input graph G, or correctly reports that G has treedepth larger than k, and OPT is the actual value of the treedepth. On the complexity side, exactly computing treedepth is NP-complete, but the known reductions do not rule out a polynomial-time approximation scheme (PTAS), and under the Exponential Time Hypothesis (ETH) only exclude a running time of $2^{o(\sqrt{n})}$ for exact algorithms.

We show that 1.0003-approximating Treedepth is NP-hard, and that exactly computing the treedepth of an n-vertex graph requires time $2^{\Omega(n)}$, unless the ETH fails. We further derive that there exist absolute constants $\delta, c > 0$ such that any $(1+\delta)$-approximation algorithm requires time $2^{\Omega(n/\log^c n)}$. We do so via a simple direct reduction from Satisfiability to Treedepth, inspired by a reduction recently designed for Treewidth [STOC '25].

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases treedepth, lower bounds, approximation

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.17

Related Version *Full Version*: https://arxiv.org/abs/2507.13818

Funding *Édouard Bonnet*: Supported by the French National Research Agency through the project TWIN-WIDTH with reference number ANR-21-CE48-0014.

1 Introduction

The treedepth $\mathrm{td}(G)$ of a graph G is the least integer k such that there is a rooted forest F of height k with same vertex set as G such that every edge of G is between two nodes in ancestor–descendant relationship in F. For every n-vertex graph G, its treedepth and treewidth, denoted by $\mathrm{tw}(G)$, are related by the inequalities $\mathrm{tw}(G)+1 \leqslant \mathrm{td}(G) \leqslant \mathrm{tw}(G)\cdot(1+\log n)$. An n-vertex path has treewidth (even pathwidth) 1, but treedepth $\Theta(\log n)$.

Treedepth comes into play in various contexts. Notably, in the sparsity theory initiated by Nešetřil and Ossona de Mendez [26], treedepth provides a characterization of classes of bounded expansion (roughly speaking, classes excluding, as subgraphs, short subdivisions of graphs of large average degree). Graph classes with bounded expansion are exactly those with so-called *low treedepth covers* (some form of cover by graphs of bounded treedepth) [21].

Graphs of bounded treewidth famously lend themselves to fixed-parameter tractable (FPT) algorithms for various NP-hard problems, with parameter the width of the computed (or given) tree-decompositions, by performing dynamic programming over these decompositions

© Édouard Bonnet, Daniel Neuen, and Marek Sokołowski;
licensed under Creative Commons License CC-BY 4.0

(see, e.g., [7, Chapter 7]). However, this method consumes essentially as much space as it takes time; in particular, most of these algorithms for NP-hard problems take exponential space in the treewidth. Bounded treedepth, in contrast, often allows for parameterized algorithms with comparable running time but using only polynomial space; see for instance [12, 20, 22, 23]. Other uses of treedepth can be found in formula complexity [18], distributed model checking [10], product structure theory [9], relation to polynomial minors [16, 14], graph circumference [6] etc. We now survey the current state of the art on computing and approximating the treedepth of an input graph – the topic of the current paper. Note that Treedepth was the selected problem for the 2020 edition of the PACE challenge [17].

The decision version of Treedepth is NP-complete [24, 3]. There is an easy algorithm that computes the treedepth of an n-vertex graph in time $O(2^n \cdot n)$, and a slightly faster exponential algorithm computing the decomposition in time $O(1.9602^n)$ [11]. An FPT (exact) algorithm to decide whether an n-vertex graph has treedepth at most k with running time $2^{O(k^2)}n$ has been established, first needing exponential space [25], and later improved to only use polynomial space [19]. More precisely, these algorithms run in time $2^{O(\mathrm{td}(G)\mathrm{tw}(G))}n$ on an n-vertex graph G. An outstanding open question is whether running time $2^{O(k)}n^{O(1)}$ can be obtained. Notably, Treewidth admits constant-approximation algorithms in this running time. In contrast to treewidth, such a result remains elusive for Treedepth. On the front of polynomial-time approximation algorithms, the best factor that currently can be achieved is $O(\mathrm{tw}(G)\log^{3/2}\mathrm{tw}(G))$ [8], hence $O(\mathrm{td}(G)\log^{3/2}\mathrm{td}(G))$.

The previously known hardness results [24, 3] are rather unsatisfactory: First of all, they did not rule out a polynomial-time approximation scheme (PTAS) for Treedepth. Furthermore, by following the reductions presented in both papers, we can only infer a $2^{\Omega(\sqrt{n})}$ lower bound for the exact variant of the problem under the Exponential Time Hypothesis (ETH).[1] Naturally, this excludes any $2^{o(\sqrt{k})}n^{O(1)}$-time parameterized exact algorithm for the problem under ETH.

Our Results. In this work, we design a simple linear reduction from Satisfiability to Treedepth. It draws inspiration from a recent similar result for Treewidth [4], and also relies on the hardness of Vertex Cover (i.e., the task of finding a smallest vertex subset hitting every edge) on tripartite graphs. Our reduction yields the following inapproximability results and computational lower bounds for Treedepth.

▶ **Theorem 1.** *It is NP-hard to 1.0003-approximate Treedepth.*

In particular, the theorem rules out a polynomial-time approximation scheme (PTAS) for Treedepth (assuming $\mathsf{P} \neq \mathsf{NP}$).

▶ **Theorem 2.** *Assuming ETH, there is some $\varepsilon > 0$ such that the treedepth of an n-vertex graph cannot be computed in time $O(2^{\varepsilon n})$.*

This also excludes $2^{o(k)}n^{O(1)}$-time parameterized exact algorithms for the problem under ETH.

In fact, we obtain that even approximating treedepth to a small constant factor requires almost exponential time.

▶ **Theorem 3.** *Assuming ETH, there exist absolute constants $\delta, \varepsilon, c > 0$ such that $(1+\delta)$-approximating the treedepth of an n-vertex graph cannot be done in time $O(2^{\varepsilon n/\log^c n})$.*

[1] The assumption that there is some $\lambda > 1$ such that n-variable 3-SAT requires time $\Omega(\lambda^n)$.

All three results follow the same reduction chain. First, we transform a Satisfiability instance into an instance of Vertex Cover on tripartite graphs. Then, we reduce the Vertex Cover problem on tripartite graphs to Treedepth. We first describe the second step of the reduction chain in Section 3. In Section 4, we then describe the first step, and deduce Theorems 1–3.

2 Preliminaries

In this work, we consider only simple undirected graphs with no self-loops. For a graph G, we define $V(G)$ to be the set of vertices, $E(G)$ to be the set of edges, and $\mathrm{cc}(G)$ to be the set of connected components of G. Also, we write $N_G(v)$ for the set of neighbors of a vertex $v \in V(G)$ in the graph G. A forest F is a graph without cycles; it is rooted if each connected component of F (a *tree*) has a *root*. The depth of a forest is then the number of vertices on the longest root-to-leaf path in F. A vertex u is an *ancestor* of v in F if u lies on the path between v and the root of the tree of F containing v; we equivalently then say that v is a *descendant* of u. In particular, every vertex is its own ancestor and descendant.

A set $S \subseteq V(G)$ is a *vertex cover* if each edge of G is incident to at least one vertex of S. The *vertex cover number* of G, denoted by $\mathrm{vc}(G)$, is the minimum cardinality of a vertex cover of G.

The treedepth of G, denoted $\mathrm{td}(G)$, is defined recursively as follows:

$$\mathrm{td}(G) = \begin{cases} 0 & \text{if } G \text{ has no vertices,} \\ \max_{H \in \mathrm{cc}(G)} \mathrm{td}(H) & \text{if } G \text{ is disconnected,} \\ \min_{v \in V(G)} 1 + \mathrm{td}(G - v) & \text{if } G \text{ is connected.} \end{cases}$$

We also use an equivalent definition of treedepth involving *elimination forests*. Here, we say that a rooted forest F is an elimination forest of G if $V(F) = V(G)$ and for every edge uv of G, vertices u and v are in the ancestor–descendant relationship in F. Then, the treedepth of G is the minimum possible depth of an elimination forest of G.

We use the following straightforward facts in our arguments:

▶ **Observation 4.** *If K is a clique in G, then in every elimination forest of G, all vertices of K are contained in a single root-to-leaf path.*

▶ **Observation 5.** *If for some nonempty $S \subseteq V(G)$, the induced subgraph $G[S]$ is connected, then in every elimination forest of G some vertex of S is an ancestor of all elements of S.*

We say that a set $X \subseteq V(G)$ of vertices of G is a *set of (false) twins* if $N_G(v) = N_G(w)$ for all $v, w \in X$. Observe that X is an independent set in this case. Every inclusion-wise minimal vertex cover of G either contains X in its entirety or is disjoint from X.

In this paper, we work with formulas with Boolean variables, say $x_1, \ldots, x_n$. A *literal* is a formula of the form x_i or $\neg x_i$. For an integer k, a Boolean formula is said to be in *k-CNF form* (or: is a *k-CNF formula*) if it is a conjunction of *clauses*: subformulas of the form $\ell_1 \vee \ldots \vee \ell_k$ for literals $\ell_1, \ldots, \ell_k$, each containing a different variable of the formula.

3 Treedepth and Vertex Cover of Tripartite Graphs

In this section, we describe a reduction from Vertex Cover on tripartite graphs to Treedepth.

We say that a graph $G = (V, E)$ is tripartite if there exists a tripartition $V = A \cup B \cup C$ such that the subgraphs of G induced by A, B, and C, respectively, are edgeless. We argue that it is possible to extend each tripartite graph G to a supergraph H by adding suitable clique gadgets so that the treedepth of H is tightly controlled by the vertex cover number of G.

▶ **Lemma 6.** *Let $G = (V, E)$ be a tripartite graph with tripartition $V = A \cup B \cup C$ such that A, B, C are nonempty and let ℓ be a positive integer such that $\ell \geqslant vc(G)$. Consider a supergraph $H = (V', E')$ of G created by adding to G three ℓ-vertex cliques K_A, K_B, K_C and three additional vertices z_A, z_B, z_C, and adding for each $X \in \{A, B, C\}$ all edges between K_X and $X \cup \{z_X\}$. That is,*

$$V' = V \cup K_A \cup K_B \cup K_C \cup \{z_A, z_B, z_C\},$$
$$E' = E \ \cup \ (A \cup \{z_A\}) \times K_A \ \cup \ (B \cup \{z_B\}) \times K_B \ \cup \ (C \cup \{z_C\}) \times K_C.$$

Then $td(H) = vc(G) + \ell + 1$.

Proof. Let S be the minimum vertex cover of G. Then $H - S$ has three connected components, namely, for each $X \in \{A, B, C\}$, the subgraph of H induced by $K_X \cup \{z_X\} \cup (X \setminus S)$. Noting that $|K_X| = \ell$ and $\{z_X\} \cup (X \setminus S)$ is an independent set in H, we have $\mathrm{td}(H[K_X \cup \{z_X\} \cup (X \setminus S)]) \leqslant |K_X| + 1 = \ell + 1$, and therefore

$$\mathrm{td}(H) \leqslant |S| + \max_{X \in \{A,B,C\}} \mathrm{td}(H[K_X \cup \{z_X\} \cup (X \setminus S)]) \leqslant \mathrm{vc}(G) + \ell + 1.$$

We now move to the lower bound on $\mathrm{td}(H)$. Aiming for contradiction, suppose that $\mathrm{td}(H) \leqslant \mathrm{vc}(G) + \ell$; then there exists an elimination forest F of H of depth at most $\mathrm{vc}(G) + \ell$.

By Observation 4, for each $X \in \{A, B, C\}$, all vertices of K_X belong to a single root-to-leaf path (possibly different for different choices of X). We can thus choose three vertices $\kappa_A \in K_A$, $\kappa_B \in K_B$, $\kappa_C \in K_C$ as the deepest vertices of the respective cliques in F.

▷ Claim 7. Let $X \in \{A, B, C\}$. Suppose that $R \subseteq V' \setminus (K_X \cup \{z_X\})$ is so that all vertices of R are ancestors of κ_X in F. Then the depth of F is at least $|R| + \ell + 1$.

Proof. All vertices of K_X are ancestors of κ_X, and z_X is either an ancestor or a descendant of κ_X. In either case, vertices of $R \cup K_X \cup \{z_X\}$ lie on a single root-to-leaf path in F, implying that the depth of F is at least $|R| + \ell + 1$. ◁

▷ Claim 8. No pair of vertices in $\{\kappa_A, \kappa_B, \kappa_C\}$ is in ancestor–descendant relationship in F.

Proof. Suppose without loss of generality that κ_A is an ancestor of κ_B. Then all vertices of K_A are ancestors of κ_B, so from Claim 7 we infer that the depth of F is at least $|K_A| + \ell + 1 = 2\ell + 1 > \mathrm{vc}(G) + \ell$; a contradiction. ◁

▷ Claim 9. Let $X, Y \in \{A, B, C\}$ with $X \neq Y$ and pick $v_X \in X$, $v_Y \in Y$ connected by an edge. Then (at least) one of the vertices v_X, v_Y is an ancestor of both κ_X and κ_Y.

Proof. Let $R = \{v_X, v_Y, \kappa_X, \kappa_Y\}$. Noting that $\kappa_X v_X v_Y \kappa_Y$ is a path in H, we have by Observation 5 that some vertex of R is an ancestor of all vertices in R. Since κ_X and κ_Y are not in the ancestor–descendant relationship (Claim 8), the claim follows. ◁

Let us now resolve a degenerate case where at least one of the sides of G (say, C) is not incident to any edge in G. Define Q to be the set of vertices of $A \cup B$ that are ancestors of both κ_A and κ_B; by Claim 9, Q is a vertex cover of G. Hence by Claim 7 for $X = A$, the depth of F is at least $|Q| + \ell + 1 > \mathrm{vc}(G) + \ell$; a contradiction.

Thus each side of G is incident to an edge of G. Therefore, we can easily verify that H is connected and so F is a single rooted tree. Let then u_{AB} be the lowest common ancestor of κ_A and κ_B in F; analogously define u_{BC} and u_{CA}. The three newly defined vertices pairwise remain in the ancestor–descendant relationship: for instance, u_{AB} and u_{BC} are both ancestors of κ_B, so one of them is an ancestor of the other. In particular, u_{AB}, u_{BC} and u_{CA} lie on a single root-to-leaf path, and (at least) one of these vertices – say, u_{AB} – is a descendant of all three. Let $Q \subseteq V$ be the set of vertices of G that are ancestors of κ_A. Applying Claim 9 multiple times, we observe that:

- for each $v_A v_B \in E(G)$ with $v_A \in A$, $v_B \in B$, we get $v_A \in Q$ or $v_B \in Q$;
- for each $v_A v_C \in E(G)$ with $v_A \in A$, $v_C \in C$, we get $v_A \in Q$ or $v_C \in Q$;
- for each $v_B v_C \in E(G)$ with $v_B \in B$, $v_C \in C$, we get that v_B or v_C is an ancestor of both κ_B and κ_C in F; hence it is also an ancestor of u_{BC}, so also an ancestor of u_{AB} and thus also κ_A. Consequently $v_B \in Q$ or $v_C \in Q$.

We conclude that Q is a vertex cover of G; and so by Claim 7, the depth of F is at least $|Q| + \ell + 1 > \mathrm{vc}(G) + \ell$; a contradiction. ◀

4 Hardness Proofs

In this section, we will prove the announced hardness results (Theorems 1–3). All reductions will start from an instance φ of Satisfiability in k-CNF form, in which every variable occurs a bounded number of times, and produce a graph (or a family of graphs) whose treedepth tightly depends on the maximum number of clauses that can be satisfied in φ. At the heart of our framework lies a construction of a tripartite graph adapted from the work of Bonnet [4], which we formally describe below.

Let φ be a k-CNF formula and p be an integer. We then define a tripartite graph $G(\varphi, p)$ as follows. Let $\gamma = 2^{k-1}p$. Define the vertex set of $G(\varphi, p)$ as $A \cup B_+ \cup B_-$, where:

- For every clause $C_i = \ell_1 \vee \ldots \vee \ell_k$ of φ and every possible choice $s_1 \in \{\ell_1, \neg\ell_1\}, \ldots, s_k \in \{\ell_k, \neg\ell_k\}$ such that $(s_1, \ldots, s_k) \neq (\neg\ell_1, \ldots, \neg\ell_k)$, we add to A a vertex $a_i(s_1, \ldots, s_k)$. Let $A(C_i)$ be the set of all vertices added to A for clause C_i. Intuitively, $A(C_i)$ contains all valuations of variables represented by the literals $\ell_1, \ldots, \ell_k$ that satisfy the clause C_i.
- For every variable x_j of φ and every $t \in [\gamma]$, we add to B_+ a vertex $b_{j,t}$ and to B_- a vertex $c_{j,t}$. Let then $B_+(x_j) = \{b_{j,1}, \ldots, b_{j,\gamma}\}$ and $B_-(x_j) = \{c_{j,1}, \ldots, c_{j,\gamma}\}$. Also, for convenience, let $B(x_j) = B_+(x_j)$ and $B(\neg x_j) = B_-(x_j)$.

We will now construct the set of edges of $G(\varphi, p)$ to ensure that every valuation of variables of φ corresponds to an inclusion-wise minimal vertex cover S that includes: (i) all vertices of A, except one vertex of $A(C_i)$ for each satisfied clause C_i, corresponding to the valuation of the variables represented by the literals of C_i, and (ii) all vertices of $B_+(x_j)$ if x_j is evaluated positively, or $B_-(x_j)$ if x_j is evaluated negatively. To this end, we construct the set E of edges of $G(\varphi, p)$ via the following process:

- Add every edge between $B(x_j)$ and $B(\neg x_j)$ for every variable x_j;
- Connect each vertex $a_i(s_1, \ldots, s_k) \in A$ with every vertex of $B(s_1) \cup \ldots \cup B(s_k)$.

This concludes the construction. We now show that the vertex cover number of $G(\varphi, p)$ is controlled by the maximum number of clauses satisfiable by φ.

▶ **Lemma 10.** *Let φ be a k-CNF formula with n variables and m clauses where every variable appears at most $2p+1$ times. Suppose also that m' is the maximum number of clauses that can be satisfied in φ. Then*

$$vc(G(\varphi,p)) = (2^k-1)m - m' + 2^{k-1}pn.$$

Proof. First, suppose that F is a valuation of variables of φ that satisfies m' clauses; we think of F as a set of literals that includes exactly one literal from $\{x_j, \neg x_j\}$ for each variable x_j. We define a set S of vertices of $G(\varphi,p)$ by including:

- every vertex of the form $a_i(s_1,\ldots,s_k) \in A$ such that $\{s_1,\ldots,s_k\} \not\subseteq F$; and
- all vertices of $B(\ell_j)$ for every literal $\ell_j \in F$.

Observe that S is a vertex cover of $G(\varphi,p)$. Indeed, every edge between B_+ and B_- is covered by S. Next, consider the edges between A and $B := B_+ \cup B_-$. Pick a vertex $a_i(s_1,\ldots,s_k) \in A \setminus S$. Every neighbor of this vertex belongs to $B(s_1) \cup \ldots \cup B(s_k)$. By construction of $S \cap A$, we have $\{s_1,\ldots,s_k\} \subseteq F$, and so $B(s_1) \cup \ldots \cup B(s_k) \subseteq S$.

Now let us determine the size of S. Since F satisfies m' clauses of φ, there exist exactly m' vertices of the form $a_i(s_1,\ldots,s_k) \in A$ such that $s_1,\ldots,s_k \in F$. These are precisely the vertices of A that do not belong to S. Therefore $|A \setminus S| = m'$ and so $|S \cap A| = (2^k-1)m - m'$. Also, $|S \cap B| = \gamma n = 2^{k-1}pn$. Therefore, $|S| = (2^k-1)m - m' + 2^{k-1}pn$.

On the other hand, suppose that S is a minimum-cardinality vertex cover of $G(\varphi,p)$; and out of those, S contains the fewest number of vertices of B. Assume that $|S| \leqslant (2^k-1)m - m'' + 2^{k-1}pn$, aiming to show that at least m'' clauses of φ can be satisfied.

For every variable x_j of φ, each of the sets $B_+(x_j)$ and $B_-(x_j)$ is a set of twins in $G(\varphi,p)$, so S either contains it fully or is disjoint from it; and moreover, S must contain at least one of the sets $B_+(x_j)$ and $B_-(x_j)$ since the vertices of both sets are connected by a complete bipartite graph. Suppose now that S contains both sets $B_+(x_j)$ and $B_-(x_j)$. Since the variable x_j appears at most $2p+1$ times in φ, we can pick a literal $\ell_j \in \{x_j, \neg x_j\}$ that appears at most p times in φ. Consider now the following set $S^\star$:

$$S^\star = (S \setminus B(\ell_j)) \cup \{a_i(s_1,\ldots,s_k) \in A \mid \ell_j \in \{s_1,\ldots,s_k\}\}.$$

Observing that $S^\star$ is formed from S by removing $B(\ell_j)$ and introducing all neighbors of $B(\ell_j)$ in $G(\varphi,p)$, we conclude that $S^\star$ is also a vertex cover of $G(\varphi,p)$. Since $|B(\ell_j)| = \gamma$, $B(\ell_j) \subseteq S$ and ℓ_j appears at most p times in φ, the cardinality of $S^\star$ is at most $|S^\star| \leqslant |S| - \gamma + 2^{k-1}p = |S|$. This contradicts the minimality of S. Therefore, for every variable x_j, S contains fully one of the sets $B_+(x_j)$, $B_-(x_j)$ and is disjoint from the other. We obtain a valuation F of φ by including in F, for every variable x_j, the literal $\ell_j \in \{x_j, \neg x_j\}$ such that $B(\ell_j) \subseteq S$. We aim to show that F satisfies at least m'' clauses of φ.

Note that $|S \cap B| = \gamma n$ and $|A| = (2^k-1)m$ and so $|A \setminus S| \geqslant m''$. Moreover, $|A(C_i) \setminus S| \leqslant 1$ for each clause C_i of φ; otherwise, we would have $a_i(s_1,\ldots,s_k), a_i(s'_1,\ldots,s'_k) \in A(C_i) \setminus S$, where $s'_j = \neg s_j$ for some $j \in [k]$. But then $a_i(s_1,\ldots,s_k)$ is connected to the vertices of $B(s_j)$ and $a_i(s'_1,\ldots,s'_k)$ is connected to the vertices of $B(s'_j)$, and by the minimality of S, either of the sets $B(s_j)$, $B(s'_j)$ is disjoint from S; a contradiction with the assumption that S is a vertex cover. Therefore, there exist at least m'' clauses C_i of φ such that $|A(C_i) \setminus S| = 1$. Observe that each such clause C_i is satisfied by F. Indeed, suppose $C_i = \ell_1 \vee \ldots \vee \ell_k$ and let $a_i(s_1,\ldots,s_k)$ be the only element of $A(C_i) \setminus S$. Then $s_j \in \{\ell_j, \neg\ell_j\}$ for each $j \in [k]$ and $(s_1,\ldots,s_k) \neq (\neg\ell_1,\ldots,\neg\ell_k)$ by the construction of $G(\varphi,p)$, and $B(s_1) \cup \ldots \cup B(s_k) \subseteq S$ by the fact that S is a vertex cover. Thus $s_1,\ldots,s_k \in F$ by the construction of F and so F satisfies φ (i.e., there is a literal $s_j \in \{s_1,\ldots,s_k\}$ such that $s_j = \ell_j$). ◀

Overall, we obtain the following reduction from Satisfiability to Vertex Cover that we record here for future reference:

▶ **Corollary 11.** *For all pairs of integers $k \geqslant 2$, $p \geqslant 1$ there exists a polynomial-time algorithm that receives as input a k-CNF formula φ with n variables and m clauses where every variable appears at most $2p+1$ times, and outputs a graph $G(\varphi, p)$ with $|V(G(\varphi,p))| = O(n)$ and the following property: If m' is the maximum number of clauses that can be satisfied in φ, then*

$$vc(G(\varphi,p)) = (2^k-1)m - m' + 2^{k-1}pn.$$

As an immediate corollary, we get that:

▶ **Corollary 12.** *For all pairs of integers $k \geqslant 2$, $p \geqslant 1$ there exists a polynomial-time algorithm that receives as input a k-CNF formula φ with n variables and m clauses where every variable appears at most $2p+1$ times, and outputs a graph $H(\varphi, p)$ with $|V(H(\varphi,p))| = O(n)$ and the following property: If m' is the maximum number of clauses that can be satisfied in φ, then*

$$td(H(\varphi,p)) = 2(2^k-1)m - m' + 2^k pn + 1.$$

Proof. Apply Lemma 6 to the graph $G(\varphi,p)$ with vertex set $A \cup B_+ \cup B_-$ and the parameter $\ell = |A \cup B_+| = (2^k-1)m + 2^{k-1}pn$ (the choice of ℓ comes from the fact that $A \cup B_+$ is a vertex cover of $G(\varphi,p)$). The value of $\mathrm{td}(H(\varphi,p))$ follows from Lemmas 6 and 10. ◀

We are now almost ready to show the approximation hardness of Treedepth (Theorem 1). We start from the following approximation hardness of the maximization variant of 2-SAT:

▶ **Theorem 13** ([2, Theorem 12])**.** *For every $\varepsilon > 0$, within the family of m-clause 2-CNF formulas where each variable appears 3 or 4 times, it is NP-hard to distinguish between the formulas where at least $(1-\varepsilon)m$ clauses are satisfiable and formulas where at most $\left(\frac{251}{252} + \varepsilon\right) m$ clauses are satisfiable.*

Proof of Theorem 1. Consider the family of 2-CNF formulas where each variable appears 3 or 4 times. In this family, every n-variable, m-clause formula satisfies $2m \geqslant 3n$, or equivalently $n \leqslant \frac{2}{3}m$. By Corollary 12, every such formula φ can be translated – in polynomial time – to a graph $H(\varphi)$ with the property that if m' is the maximum number of clauses satisfiable in φ, then $\mathrm{td}(H(\varphi)) = 6m - m' + 8n + 1$.

Now let $\delta < \frac{1}{2604}$ be fixed. Then there is some $\varepsilon > 0$ such that, for large enough m and $n \leqslant \frac{2}{3}m$,

$$(6m - (1-\varepsilon)m + 8n + 1)(1+\delta) < 6m - \left(\frac{251}{252} + \varepsilon\right) m + 8n + 1.$$

So, letting $k = 6m - (1-\varepsilon)m + 8n$, distinguishing between formulas φ where at least $(1-\varepsilon)m$ clauses are satisfiable and formulas where at most $\left(\frac{251}{252} + \varepsilon\right) m$ clauses are satisfiable reduces to distinguishing between graphs of treedepth at most k and those of treedepth at least $(1+\delta)k$. Hence, we obtain hardness by Theorem 13. ◀

We move to the hardness results under the Exponential Time Hypothesis (ETH); both presented proofs invoke the Sparsification Lemma of Impagliazzo, Paturi, and Zane [15]. We use this lemma in the following form:

▶ **Lemma 14** (Sparsification Lemma [15])**.** *For every $0 < \varepsilon' < 0.1$ there exists a constant $B > 0$ such that a 3-CNF formula φ with n' variables can be transformed in time $O(2^{\varepsilon' n'})$ into $s \leqslant 2^{\varepsilon' n'}$ 3-CNF formulas $\varphi_1, \ldots, \varphi_s$ with the same set of variables such that*

- **(i)** *each variable appears at most B times in each formula φ_i, and*
- **(ii)** *φ is satisfiable if and only if one of the formulas φ_i is satisfiable.*

The Sparsification Lemma, when combined with ETH and Corollary 12, yields a straightforward proof of Theorem 2:

Proof of Theorem 2. Lemma 14 together with ETH implies that, for some absolute constants $\varepsilon', B > 0$, no algorithm solves 3-CNF instances of the satisfiability problem on n' variables, with each variable appearing at most B times, in time $O(2^{\varepsilon' n'})$. Given such an instance φ with n' variables and m' clauses, we use Corollary 12 with $k = 3$, $p = \left\lfloor \frac{B}{2} \right\rfloor$ to transform it, in polynomial time, into a graph $H(\varphi)$ with $|V(H(\varphi))| \leqslant c \cdot n'$ for some fixed constant c (where n' is sufficiently large). Then φ is satisfiable if and only if $\mathrm{td}(H(\varphi)) = (2^{k+1}-3)m' + 2^k p n' + 1$. Therefore, under ETH, the treedepth of an n-vertex graph cannot be computed in time $O(2^{\varepsilon n})$ where $\varepsilon \coloneqq \varepsilon'/(2c)$. ◀

The approximation time complexity lower bound (Theorem 3) requires a bit more work:

Proof of Theorem 3. Assuming ETH, there is some $\lambda > 0$ such that n'-variable 3-SAT cannot be solved in time $O(2^{\lambda n'})$. Pick $0 < \varepsilon' < \min\{0.1, \frac{1}{2}\lambda\}$ and invoke the Sparsification Lemma (Lemma 14). Let the 3-CNF formulas φ and $\varphi_1, \ldots, \varphi_s$ be as in the statement of the lemma. Combining the Sparsification Lemma with a polynomial-time Satisfiability inapproximability framework of Håstad [13] and a quasi-linear-size construction of polynomially-checkable proofs (PCP) by Bafna, Minzer, Vyas, and Yun [1], we find absolute constants $0 < \alpha_1 < \alpha_2 < 1$ and $B^\star, c > 0$ such that we can translate each formula φ_i in polynomial time to a 3-CNF formula $\varphi_i^\star$ with $n^\star \leqslant n' \log^c n'$ variables, each appearing at most $B^\star$ times, with the following property: If $\varphi_i^\star$ has $m^\star$ clauses, then:

- If φ_i is satisfiable, then at least $\alpha_2 m^\star$ clauses of $\varphi_i^\star$ can be satisfied.
- If φ_i is not satisfiable, then at most $\alpha_1 m^\star$ clauses of $\varphi_i^\star$ can be satisfied.

(This reduction is standard; see [5, Lemma 2] for the proof of an analogous result under the assumption of the existence of a linear-size PCP construction.) Chaining this result with Corollary 12, we find absolute constants $0 < \beta_1 < \beta_2 < 1$ and a polynomial-time algorithm that transforms each $\varphi_i^\star$ into a graph H_i with at most $dn^\star$ vertices, where d is a fixed constant, such that:

- If φ_i is satisfiable, then $\mathrm{td}(H_i) \leqslant \beta_1 |V(H_i)|$.
- If φ_i is not satisfiable, then $\mathrm{td}(H_i) > \beta_2 |V(H_i)|$.

Now, let $\varepsilon \coloneqq \varepsilon'/d$ and $\delta = \beta_2/\beta_1 - 1$. Then, supposing that a $O(2^{\varepsilon n/\log^c n})$-time $(1+\delta)$-approximation algorithm for treedepth existed, the satisfiability of φ could be decided in time

$$O(2^{\varepsilon' n'}) + s \cdot ((n')^{O(1)} + O(2^{\varepsilon d n^\star / \log^c n^\star})) \leqslant O(2^{\varepsilon' n'}) + 2^{\varepsilon' n'} \cdot ((n')^{O(1)} + O(2^{\varepsilon' n'})) = O(2^{\lambda n'}),$$

thus refuting the ETH. ◀

References

1 Mitali Bafna, Dor Minzer, Nikhil Vyas, and Zhiwei Yun. Quasi-linear size PCPs with small soundness from HDX. In Michal Koucký and Nikhil Bansal, editors, *Proceedings of the 57th Annual ACM Symposium on Theory of Computing, STOC 2025, Prague, Czechia, June 23-27, 2025*, pages 45–53. ACM, 2025. doi:10.1145/3717823.3718197.

2 Piotr Berman and Marek Karpinski. Improved approximation lower bounds on small occurrence optimization. *Electron. Colloquium Comput. Complex.*, TR03-008, 2003. URL: https://eccc.weizmann.ac.il/eccc-reports/2003/TR03-008/index.html.

3 Hans L. Bodlaender, Jitender S. Deogun, Klaus Jansen, Ton Kloks, Dieter Kratsch, Haiko Müller, and Zsolt Tuza. Rankings of graphs. *SIAM J. Discret. Math.*, 11(1):168–181, 1998. doi:10.1137/S0895480195282550.

4 Édouard Bonnet. Treewidth inapproximability and tight ETH lower bound. In Michal Koucký and Nikhil Bansal, editors, *Proceedings of the 57th Annual ACM Symposium on Theory of Computing, STOC 2025, Prague, Czechia, June 23-27, 2025*, pages 2130–2135. ACM, 2025. doi:10.1145/3717823.3718117.

5 Édouard Bonnet, Bruno Escoffier, Eun Jung Kim, and Vangelis Th. Paschos. On Subexponential and FPT-Time Inapproximability. *Algorithmica*, 71(3):541–565, 2015. doi:10.1007/s00453-014-9889-1.

6 Marcin Briański, Gwenaël Joret, Konrad Majewski, Piotr Micek, Michał T. Seweryn, and Roohani Sharma. Treedepth vs circumference. *Comb.*, 43(4):659–664, 2023. doi:10.1007/s00493-023-00028-5.

7 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.

8 Wojciech Czerwiński, Wojciech Nadara, and Marcin Pilipczuk. Improved bounds for the excluded-minor approximation of treedepth. 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 34:1–34:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ESA.2019.34.

9 Vida Dujmović, Robert Hickingbotham, Gwenaël Joret, Piotr Micek, Pat Morin, and David R. Wood. The excluded tree minor theorem revisited. *Comb. Probab. Comput.*, 33(1):85–90, 2024. doi:10.1017/s0963548323000275.

10 Fedor V. Fomin, Pierre Fraigniaud, Pedro Montealegre, Ivan Rapaport, and Ioan Todinca. Distributed model checking on graphs of bounded treedepth. In Dan Alistarh, editor, *38th International Symposium on Distributed Computing, DISC 2024, October 28 to November 1, 2024, Madrid, Spain*, volume 319 of *LIPIcs*, pages 25:1–25:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.DISC.2024.25.

11 Fedor V. Fomin, Archontia C. Giannopoulou, and Michał Pilipczuk. Computing tree-depth faster than 2^n. *Algorithmica*, 73(1):202–216, 2015. doi:10.1007/s00453-014-9914-4.

12 Martin Fürer and Huiwen Yu. Space saving by dynamic algebraization based on tree-depth. *Theory Comput. Syst.*, 61(2):283–304, 2017. doi:10.1007/s00224-017-9751-3.

13 Johan Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, 2001. doi:10.1145/502090.502098.

14 Meike Hatzel, Gwenaël Joret, Piotr Micek, Marcin Pilipczuk, Torsten Ueckerdt, and Bartosz Walczak. Tight bound on treedepth in terms of pathwidth and longest path. *Comb.*, 44(2):417–427, 2024. doi:10.1007/s00493-023-00077-w.

15 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.

16 Ken-ichi Kawarabayashi and Benjamin Rossman. A polynomial excluded-minor approximation of treedepth. *J. Eur. Math. Soc. (JEMS)*, 24(4):1449–1470, 2022. doi:10.4171/JEMS/1133.

17 Lukasz Kowalik, Marcin Mucha, Wojciech Nadara, Marcin Pilipczuk, Manuel Sorge, and Piotr Wygocki. The PACE 2020 parameterized algorithms and computational experiments challenge: Treedepth. 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 37:1–37:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.IPEC.2020.37.

18 Deepanshu Kush and Benjamin Rossman. Tree-depth and the formula complexity of subgraph isomorphism. *SIAM J. Comput.*, 52(1):273–325, 2023. doi:10.1137/20m1372925.

19 Wojciech Nadara, Michał Pilipczuk, and Marcin Smulewicz. Computing treedepth in polynomial space and linear FPT time. 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 79:1–79:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ESA.2022.79.

20 Jesper Nederlof, Michał Pilipczuk, Céline M. F. Swennenhuis, and Karol Węgrzycki. Hamiltonian cycle parameterized by treedepth in single exponential time and polynomial space. *SIAM J. Discret. Math.*, 37(3):1566–1586, 2023. doi:10.1137/22m1518943.

21 Jaroslav Nešetřil and Patrice Ossona de Mendez. Grad and classes with bounded expansion i. decompositions. *Eur. J. Comb.*, 29(3):760–776, 2008. doi:10.1016/j.ejc.2006.07.013.

22 Michał Pilipczuk and Sebastian Siebertz. Polynomial bounds for centered colorings on proper minor-closed graph classes. *J. Comb. Theory B*, 151:111–147, 2021. doi:10.1016/j.jctb.2021.06.002.

23 Michał Pilipczuk and Marcin Wrochna. On space efficiency of algorithms working on structural decompositions of graphs. *ACM Trans. Comput. Theory*, 9(4):18:1–18:36, 2018. doi:10.1145/3154856.

24 Alex Pothen. The complexity of optimal elimination trees. *Technical Report, Pennsylvania State University*, 1988. URL: https://www.cs.purdue.edu/homes/apothen/Papers/shortest-etree1988.pdf.

25 Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, and Somnath Sikdar. A Faster Parameterized Algorithm for Treedepth. 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 I*, volume 8572 of *Lecture Notes in Computer Science*, pages 931–942. Springer, 2014. doi:10.1007/978-3-662-43948-7_77.

26 Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity - Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012. doi:10.1007/978-3-642-27875-4.

Bridging Treewidth and Clique-Width via Cograph-Modular-Treewidth

Václav Blažej
University of Warsaw, Poland

Satyabrata Jana
University of Warwick, Coventry, UK

M. S. Ramanujan
University of Warwick, Coventry, UK

Peter Strulo
University of Warwick, Coventry, UK

Abstract

Many classical graph problems – such as Max Cut, Chromatic Number, Edge Dominating Set, and Hamiltonian Cycle – are polynomial-time solvable on cographs, fixed-parameter tractable (FPT) when parameterized by treewidth, but W[1]-hard when parameterized by clique-width. In contrast, Graph Isomorphism is FPT parameterized by treewidth, but for clique-width it is known to be in XP; whether it is FPT or W[1]-hard is open.

This reveals a sharp tractability gap between treewidth and clique-width. In this work, we propose a new structural graph parameter, $\mathcal{C}$-modular-treewidth, which lies between treewidth and clique-width. The parameter leverages modular decomposition and restricts modules to induce graphs from a fixed class $\mathcal{C}$ (e.g., cographs or edgeless graphs). By exploiting true and false twins – a hallmark of cograph-like structure – our parameter allows the design of efficient algorithms for several hard problems beyond the reach of treewidth-based methods. In this work, we show that $\mathcal{C}$-modular-treewidth enables efficient solutions under suitable choices of $\mathcal{C}$, opening a new pathway in the parameterized complexity landscape between treewidth and clique-width. In particular we show that

- When parameterized by cograph-modular-treewidth, Isomorphism admits an FPT algorithm, whereas Chromatic Number remains W[1]-hard.
- When parameterized by independent-modular-treewidth, Hamiltonian Cycle and Edge Dominating Set remain W[1]-hard.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Treewidth, Clique-width, Cograph, FPT, W[1]-hard

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.18

Funding *Václav Blažej*: Supported by project BOBR that received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 948057).
Satyabrata Jana: Supported by the Engineering and Physical Sciences Research Council (grant number EP/V044621/1).
M. S. Ramanujan: Supported by the Engineering and Physical Sciences Research Council (grant number EP/V044621/1).
Peter Strulo: Supported by the Engineering and Physical Sciences Research Council (grant number EP/V044621/1).

© Václav Blažej, Satyabrata Jana, M. S. Ramanujan, and Peter Strulo;
licensed under Creative Commons License CC-BY 4.0

1 Introduction

The field of parameterized complexity offers powerful tools for tackling computationally hard problems by identifying structural parameters that govern the complexity of input instances. Among these parameters, *treewidth* has played a central role due to its ability to capture the tree-like structure of graphs. Many graph problems that are NP-hard in general become tractable when parameterized by treewidth. This is because a wide range of problems that are solvable in polynomial time on trees can also be efficiently solved on graphs of bounded treewidth [5, 15, 16, 18, 19, 28, 32]. A foundational result in this area is that any graph problem expressible in monadic second-order logic (MSO) can be solved in linear time on graphs of bounded treewidth [2, 10, 11, 13]. This result captures a large family of natural problem such as Hamiltonian Cycle, and optimization versions of the problems like Vertex Cover, Dominating Set, Feedback Vertex Set, Max Cut, Chromatic Number and Edge Dominating Set. For a comprehensive survey on treewidth and its algorithmic implications, we refer the reader to Bodlaender [3]. At the other end of the spectrum lies *clique-width*, a more general graph parameter introduced by Courcelle and Olariu [14]. Clique-width generalizes treewidth in the sense that every graph class of bounded treewidth also has bounded clique-width [8]. Importantly, for graph problems expressible in monadic second-order logic without edge set quantification (so-called MS_1 logic), Courcelle, Makowsky, and Rotics [12] showed that such problems are fixed-parameter tractable (FPT) when parameterized by clique-width. However, this generality comes at a cost. Many problems that are tractable for bounded treewidth become W[1]-hard or even para-NP-hard when parameterized by clique-width [20, 21, 22]. This reveals a crucial algorithmic gap and motivates the need for *intermediate parameters* – those that extend beyond treewidth but remain more structured than clique-width to retain algorithmic tractability.

In this work, we introduce such an intermediate parameter: *$\mathcal{C}$-modular-treewidth*. This parameter lies strictly between treewidth and clique-width and is inspired by the classical notion of modular decomposition, which partitions a graph into *modules* – vertex sets that share uniform adjacency to the rest of the graph. The idea is to restrict the induced subgraphs formed by these modules to belong to a fixed class $\mathcal{C}$, thereby allowing controlled generalization beyond treewidth. We consider the following five problems in our framework.

Isomorphism

Input: Two graphs G_1 and G_2.

Question: Are G_1 and G_2 isomorphic, i.e., is there a bijection $\phi\colon V(G_1) \to V(G_2)$ such that $uv \in E(G_1)$ if and only if $\phi(u)\phi(v) \in E(G_2)$?

Max Cut

Input: A graph G, a positive integer r.

Question: Can we partition $V(G)$ into two subsets such that at least r edges have endpoints in both subsets?

Chromatic Number

Input: A graph G, a positive integer r.

Question: Can we color the vertices of G using at most r colors such that no adjacent vertices receive the same color?

Edge Dominating Set

Input: A graph G, a positive integer r.

Question: Is there an edge subset $X \subseteq E(G)$ of size at most r such that every edge in G is either in X or adjacent to an edge in X?

Hamiltonian Cycle
Input: A graph G.
Question: Is there a cycle that passes through every vertex of G exactly once?

A central case of interest is when $\mathcal{C}$ consists of *cographs* (i.e., P_4-free graphs) or *edgeless graphs*. Cographs, defined recursively via disjoint union and join operations, are well-known for their structural simplicity: they have clique-width at most two, admit cotree representations, and are closed under modular decomposition. Many difficult problems become polynomial-time solvable on cographs, including Chromatic Number and Hamiltonian Cycle. A key feature of cographs is the occurrences of *twin vertices* – pairs of vertices with identical neighborhoods (true twins if adjacent, false twins if not). These twins play a foundational role in modular decomposition and in our parameter. By leveraging the presence of twin sets, we define a decomposition that generalizes tree decompositions while enforcing that the modules belong to a restricted class $\mathcal{C}$.

Motivation and Applications. Our motivation is to bridge the algorithmic tractability gap between treewidth and clique-width. The same question was asked by Sæther and Telle [30] who introduced the graph parameter *sm-width*, which lies between treewidth and clique-width. Notably, some graph classes with unbounded treewidth, like distance-hereditary graphs, have bounded sm-width. In [30] the authors showed that MaxCut, Chromatic Number, Hamiltonian Cycle, and Edge Dominating Set are FPT when parameterized by sm-width.

Another parameter between treewidth and clique-width is called *modular-treewidth* and was introduced by Bodlaender and Jansen [4]. Notably Lampis [25] studied Chromatic Number parameterized by modular-treewidth (mtw), showing an algorithm that decides k-coloring in $\mathcal{O}^*(\binom{k}{\lfloor k/2 \rfloor}^{\texttt{mtw}})$ time; this is FPT when parameterized by $k + \texttt{mtw}$. Hegerfeld and Kratsch [23] recently studied connectivity problems on modular-width. Notably, they define *twinclass-treewidth* which is equivalent to ours $\mathcal{C}$-modular treewidth when $\mathcal{C}$ is set to be (clique $\cup$ edgeless), i.e., the class of all cliques and edgeless graphs. Several papers [25, 27, 29] use the name modular-treewidth to refer twinclass-treewidth.

Many natural problems are polynomial-time solvable on cographs, FPT when parameterized by treewidth, yet become intractable (e.g., W[1]-hard) under clique-width. Examples include Max Cut, Chromatic Number, Edge Dominating Set, Hamiltonian Cycle. Although Isomorphism is FPT when parameterized by treewidth [26], for clique-width it is only known to be in XP [22] and whether it is FPT or W[1]-hard remains open.

W[1]-hardness reductions for many of these problems exhibit unbounded treewidth due to a simple obstruction, e.g. big cliques or bi-cliques, which additionally form modules in the constructed graph. This, together with their tractability on treewidth, raises the question of whether simple modules can be exploited to design FPT algorithms.

Our Results. In the following sections, we formally define $\mathcal{C}$-modular-treewidth, analyze its structural and algorithmic properties, and show its applicability to the above problems. Our results contribute to the broader goal of identifying and utilizing structural graph parameters that support efficient algorithms, lying between the extremes of treewidth and clique-width.

In Section 2.1 we clarify relations to other parameters and why results for modular-treewidth extend to our parameters. In particular, note that for Chromatic Number the number of colors is naturally bounded for graphs of treewidth, which is also true for independent-modular-treewidth. On the other hand, the number of colors is unbounded in cographs, and we support untractibility in this case by showing coloring is W[1]-hard by cograph-modular-treewidth.

Table 1 Overview of results for the $\mathcal{C}$-modular-treewidth (-mt) parameter variants.

problem↓ parameter→	treewidth	independent-mt	cograph-mt	clique-width
Isomorphism	FPT [26]	FPT	FPT [Sec. 4]	XP [22]
Max Cut	FPT [4]	FPT	FPT [4]	W[1]-hard [21]
Chromatic Number	FPT [20]	FPT [25]	W[1]-h. [Sec. 5]	W[1]-hard [20]
Edge Dominating Set	FPT [20]	W[1]-h. [Sec. 6]	W[1]-hard	W[1]-h. [20, 21]
Hamiltonian Cycle	FPT [20]	W[1]-h. [Sec. 7]	W[1]-hard	W[1]-hard [20]

To simplify the notation in this paper, we employ the following convention while discussing problems. We use the following shorthand for width parameters: `tw` for treewidth, `cw` for clique-width, `imtw` for independent-modular-treewidth (i.e., $\mathcal{C}$-modular-treewidth with $\mathcal{C}$ as edgeless graphs), and `cmtw` for cograph-modular-treewidth (i.e., $\mathcal{C}$-modular-treewidth with $\mathcal{C}$ as cographs). For a graph problem Π and a width parameter $\mu \in \{\texttt{tw}, \texttt{cw}, \texttt{imtw}, \texttt{cmtw}\}$, when we represent it as Π/μ it indicates that the problem Π is considered with parameter μ. For example, Isomorphism/`tw` indicates Isomorphism parameterized by treewidth.

2 Preliminaries

We consider undirected unweighted simple graphs, unless stated otherwise. A graph G has n vertices $V(G)$ and m edges $E(G)$. For a set of vertices $X \subseteq V(G)$ we use $\binom{X}{2}$ for $\{uv \mid u, v \in X, u \neq v\}$, i.e., all the possible edges with both endpoints in X. For adjacency we use $u \sim v \iff uv \in E(G)$; similarly, $u \nsim v$ denotes $uv \notin E(G)$. If one side (or both sides) of an adjacency relation contains a set $u \sim X$ we mean $u \sim x$ for every $x \in X$, similarly for $u \nsim X$. We use $\sim_G$ if the graph of the adjacency is not clear from the context. Let $G[X]$ for $X \subseteq V(G)$ denote the graph induced on vertices X, i.e., $G[X] = (X, E(G) \cap \binom{X}{2})$. Open neighborhood of u is denoted $N(u)$ and its closed neighborhood $N[u] = \{u\} \cup N(u)$. Two vertices u, v are called *true twins* if $N[u] = N[v]$ (implying $u \sim v$), and *false twins* if $N(u) = N(v)$ (implying $u \nsim v$). We use $G_1 \approx G_2$ to denote that G_1 is isomorphic to G_2.

▶ **Definition 1** (Module)**.** For a graph G, a subset of vertices $M \subseteq V(G)$ is said to be a *module* in G if for each $v \notin M$ either $v \sim M$ or $v \nsim M$.

Treewidth. A *tree decomposition* of an undirected graph G is a pair $(T, \{X_t\}_{t \in V(T)})$ where T is a tree and $X_t \subseteq V(G)$ such that (i) for all edges $uv \in E(G)$ there exists a node $t \in V(T)$ such that $\{u, v\} \subseteq X_t$ and (ii) for all $v \in V(G)$ the subgraph induced by $\{t \in V(T) \mid v \in X_t\}$ is a non-empty tree. The *width* of a tree decomposition is $\max_{t \in V(T)} |X_t| - 1$. The *treewidth* of G is the minimum width of a tree decomposition of G.

▶ **Definition 2** ($\mathcal{C}$-modular tree-decomposition)**.** *$\mathcal{C}$-modular tree-decomposition* of a graph G is a triple $(H, p, \mathcal{T})$ where H is a graph, $p\colon V(G) \to V(H)$ is a surjective function such that $u \sim_H v \Leftrightarrow p^{-1}(u) \sim_G p^{-1}(v)$ and for each $u \in V(H)$ we have $G[p^{-1}(u)] \in \mathcal{C}$, and $\mathcal{T}$ is a tree-decomposition of H. Width of a $\mathcal{C}$-modular tree-decomposition is defined as the width of the tree decomposition $\mathcal{T}$.

Observe that Definition 2 is equivalent to partitioning G into modules with each part inducing a graph from $\mathcal{C}$.

▶ **Definition 3** ($\mathcal{C}$-modular-treewidth)**.** *$\mathcal{C}$-modular-treewidth* ($\mathcal{C}$-`mtw`) of a graph G is the minimum k such that G has a $\mathcal{C}$-modular tree-decomposition of width k.

In other words, by the above definition we can construct G by taking a graph H of bounded treewidth and replace every vertex $u \in V(H)$ with a graph $p^{-1}(u) \in \mathcal{C}$, making all the replacement vertices adjacent to the prior neighbors of u, see Figure 1. Observe that for each $v \in V(H)$ every $p^{-1}(v)$ forms a module of G. We call H the *module graph* and the function p the *replacement function*.

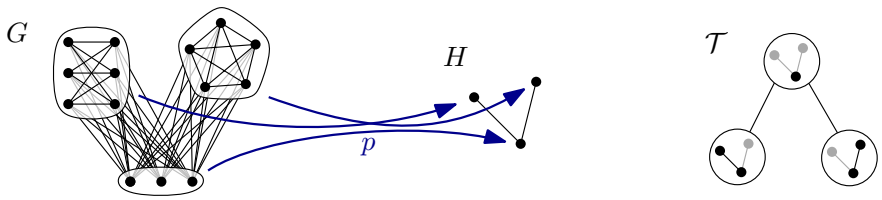


Figure 1 Example of a graph G which has $\mathcal{C}$-modular-treewidth 1 for $\mathcal{C}$ containing complete graphs, independent sets, and complete bipartite graphs. $\mathcal{C}$-modular-treewidth is 1 because H has a tree-decomposition $\mathcal{T}$ of width 1.

Note that if $\mathcal{C}$ contains a single-vertex graph then every graph has a $\mathcal{C}$-modular tree-decomposition as we can set $H = G$, p to be identity and $\mathcal{T}$ to be the tree-decomposition of G.

We now show basic properties of this general definition.

▶ **Observation 4.** *$\mathcal{S}$-modular-treewidth, with $\mathcal{S}$ being a class that consists of only one graph that contains a single vertex, is equivalent to treewidth.*

Proof. With the class $\mathcal{S}$ being this restricted the replacement function p cannot change the graph at all, hence, $G \approx H$ and we conclude that G has treewidth k if and only if it has $\mathcal{S}$-modular-treewidth k. ◀

▶ **Observation 5.** *Any graph $G \in \mathcal{C}$ has $\mathcal{C}$-modular-treewidth 0.*

Proof. As G is from the class $\mathcal{C}$ it follows that it can be created from a single vertex u by replacing u with G. The graph with a single vertex has treewidth 0. ◀

▶ **Corollary 6.** *Every edgeless graph G has $\texttt{imtw}(G) = 0$. Every cograph H has $\texttt{cmtw}(H) = 0$.*

▶ **Lemma 7.** *For a graph G and graph classes $\mathcal{C}$ and $\mathcal{D}$ such that $\mathcal{C} \subseteq \mathcal{D}$ we have $\mathcal{D}$-modular-treewidth of G is at most $\mathcal{C}$-modular-treewidth of G.*

Proof. Consider a $\mathcal{C}$-modular tree-decomposition $(H, p, \mathcal{T})$, then $(H, p, \mathcal{T})$ is also a $\mathcal{D}$-modular tree-decomposition because any part of p induces a graph of $\mathcal{C}$ which is a subset of $\mathcal{D}$; the other conditions remain unchanged and still hold. ◀

It is clear that the usefulness of $\mathcal{C}$-modular-treewidth depends heavily on the chosen graph class $\mathcal{C}$. One challenge with general graph classes is to be able to find a $\mathcal{C}$-modular tree-decomposition; one central class in our work is the class of cograph and to overcome this challenge we exploit the nice structure implied by the presence of twins in the class of cographs.

▶ **Definition 8** (Disjoint union, complement, join)**.** *Let $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$ be two distinct graphs. The* disjoint union *of G_1 and G_2 is the graph $G_1 \cup G_2 = (V_1 \cup V_2, E_1 \cup E_2)$. The* complement *of G_1 is $\bar{G}_1 = (V_1, \binom{V_1}{2} \setminus E_1)$. The* join *of graphs G_1 and G_2 is $(V_1 \cup V_2, E_1 \cup E_2 \cup \{uv \mid u \in V_1, v \in V_2\})$.*

▶ **Definition 9** (Cographs)**.** Cographs (complement reducible graphs) are defined recursively as follows: 1) a single vertex is a cograph; 2) the disjoint union of cographs is a cograph; 3) the join of cographs is a cograph.

As one can replace join with a sequence of 3 operations: complement, disjoint union, complement; it is not difficult to see that replacing (3) with taking complement of a cograph yields an equivalent (original) definition.

▶ **Definition 10** (Cotree [9])**.** *Cotree* $T(G)$ of a cograph G is a rooted tree with unordered children whose set of leaves is $V(G)$, each internal vertex (except possibly root) has degree at least 2, each internal vertex is labeled by its depth plus one modulo 2, and for each $u, v \in V(G)$ we have $u \sim v$ if and only if the least common ancestor of the leaves that represent u and v is labeled with 1.

It was observed in [9] that each cograph has a unique representation by a cotree. We later use the fact that there is an algorithm to retrieve cotree and that tree isomorphism is a solved problem.

For a graph G, we denote its treewidth $\mathtt{tw}(G)$, clique-width $\mathtt{cw}(G)$, cograph-modular-treewidth $\mathtt{cmtw}(G)$, and independent-modular-treewidth $\mathtt{imtw}(G)$.

Parameterized Complexity. We refer to [15] for an introduction to the area. A parameterized problem (x, k) where x is a string over Σ and a parameter $k \in \mathbb{N}$ is fixed-parameter tractable (or FPT) if it can be solved in $f(k) \cdot |x|^{\mathcal{O}(1)}$ time for some computable function f. The standard notion of fixed-parameter intractability is W[1]-hardness.

2.1 $\mathcal{C}$-modular-treewidth and other parameters

In this short section, we place the introduced parameters within the context of other parameters. Parameters we use solely in this section are modular-treewidth $\mathtt{mtw}(G)$, modular-width $\mathtt{mw}(G)$, and split-matching-width (sm-width) $\mathtt{smw}(G)$. See the overview in Figure 2.

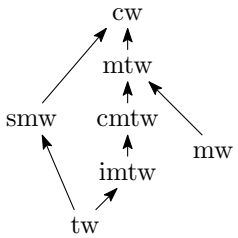


Figure 2 Parameter hierarchy for treewidth (tw), sm-width (smw), modular-treewidth (mtw), modular-width (mw), clique-width (cw), independent-modular-treewidth (imtw), and cograph-modular-treewidth (cmtw). Arrow from A to B represents that if $A(G)$ is bounded, then $B(G)$ is bounded.

▶ **Lemma 11.** *For any graph G we have* $\mathtt{cmtw}(G) \leq \mathtt{imtw}(G) \leq \mathtt{tw}(G)$.

Proof. Let us have graph classes: edgeless $\mathcal{I}$, cographs $\mathcal{R}$, graph with one vertex $\mathcal{S}$. Due to Observation 4 we know that treewidth is equivalent to $\mathcal{S}$-modular-treewidth. As $\mathcal{S} \subseteq \mathcal{I} \subseteq \mathcal{R}$ we can use Lemma 7 to conclude that $\mathtt{cmtw}(G) \leq \mathtt{imtw}(G) \leq \mathtt{tw}(G)$. ◀

▶ **Lemma 12.** *For any graph G we have* $\mathtt{mtw}(G) \leq \mathtt{cmtw}(G)$.

Proof. Let $k = \texttt{cmtw}(G)$. Let $(H, p, \mathcal{T})$ be the cograph-modular tree-decomposition that witnesses cograph-modular-treewidth of G is k. Observe, that one can alter this decomposition by taking modular decomposition of every co-graph and placing it in the stead of the cograph. This alteration results in a modular decomposition witnessing that $\texttt{mtw}(G) \leq k$ because every cograph will be decomposed into its cotree sturcture, resulting in tree nodes that are only cliques or independent sets that do not increase modular-treewidth of the decomposition. ◀

▶ **Lemma 13.** `mw` *is incomparable to* `cmtw`*,* `imtw`*, and* `tw`*.*

Proof. Note that due to Lemma 11 it suffices to show a graph class with bounded `tw` and unbounded `mw`, and another graph class with bounded `mw` and unbounded `cmtw`. We claim that the first graph class is a class of all paths – note that its treewidth is 1 as all paths are also trees; the modular-width cannot decompose any path because a path graph contains only trivial modules.

The second graph class we create inductively as follows. Let G_0 be a path on 4 vertices. Let G_{i+1} consists of 4 modules, each containing exactly G_i, and let the quotient graph over these modules be a path on 4 vertices. Let $\mathcal{G} = \cup_{i=0}^{\infty}\{G_i\}$. By design of $\mathcal{G}$, each graph $G \in \mathcal{G}$ has $\texttt{mw}(G) \leq 4$. On the other hand, notice that G contains no true nor false twins. We will later see in Lemma 19 that every non-trivial module of a cograph-modular tree-decomposition contains twins. Therefore, the decomposition of G is trivial (each module contains a single vertex) so the quotient graph contains a complete bipartite graph of unbounded size, hence, $\texttt{cmtw}(\mathcal{G})$ is unbounded. ◀

Later, we show that hardness of Edge Dominating Set on independent-modular-treewidth; this problem is FPT on sm-width, hence we conclude the following.

▶ **Corollary 14.** *Unless* FPT=W[1]*, graph classes with bounded* `imtw` *do not necessarily have bounded* `smw`*.*

We did not find any reference for the relation between `smw` and `mtw`, nor a result that would prove incomparability of `smw` to our parameters. The above corollary shows one incomparability direction but it still remains to determine whether bounded `smw` implies bounded `mtw`, `cmtw`, or even `imtw`.

▶ **Lemma 15.** *There is a graph class with bounded* `smw` *and unbounded* `mtw`*.*

Proof. Consider a graph class $\mathcal{G}$ that for each $n \geq 3$ contains a graph consisting of a clique K_n where each vertex of the clique $u \in V(K_n)$ has a private pendant leaf ℓ_u. We claim that $\mathcal{G}$ has bounded `smw` but unbounded `mtw`. To show bounded `smw` it is not hard to find embedding of $V(G)$ into leaves of a ternary tree such that each subgraph induced by edges between parts of a bipartition that is implied by every edge of the ternary tree forms a complete bipartite graph. Unboundedness of `mtw` comes from the fact that every $G \in \mathcal{G}$ contains no non-trivial modules and so the first quotient graph must contain a clique K_n which lower bounds treewith by $n - 1$. ◀

Due to Lemma 11, Lemma 15, and Corollary 14 we get the following corollary.

▶ **Corollary 16.** *Unless* FPT=W[1] *we have that* `smw` *is incomparable to* `mtw`*,* `cmtw`*, and* `imtw`*.*

As `mtw` is defined as treewidth of quotient graphs within primal nodes of modular decomposition of the graph whereas `mw` is simply the maximum number of vertices within such nodes, implying $\texttt{mtw}(G) \leq \texttt{mw}(G)$ for any graph G.

The fact that bounded smw implies bounded cw is a combination of several facts: 1) clique-width is bounded by treewidth $\texttt{cw}(G) \leq 2^{\texttt{tw}(G)+1} + 1$ [14] – as each quotient graph of primal nodes has buonded treewidth one knows that there is an expression witnessing bounded clique-width of the quotient graph; 2) if we have an expression for a module, then we can change all vertices of the module to a single color and assume there is a single vertex instead – hence, we can easily attach expressions of modules to the expression of the parent quotient graph. Applying this recursively quields an expression of bounded clique-width.

3 Algorithm to find decompositions for cmtw and imtw

The usefulness of a parameter towards designing FPT algorithms is significantly impacted by our ability to retrieve a decomposition that witnesses that the parameter is bounded.

We first show an auxiliary lemma that shows structure of module graph of the decompositions. Then we focus on obtaining H and p of $\mathcal{C}$-modular tree-decompositions. We conclude with noting that $\mathcal{T}$ of H can be found by showing how to obtain independent-modular tree-decomposition, which is quite simple, and then move to cograph-modular tree-decomposition for which we will need a number of auxiliary notions.

The below approaches are presented in order to have a self-contained contribution. We believe that using the linear-time algorithm for modular decomposition [31, 7] one can obtain independent- and cograph-modular-treewidth in linear time.

▶ **Lemma 17.** *For a graph G that has $\mathcal{C}$-modular-treewidth k there exists a $\mathcal{C}$-modular tree-decomposition $(H, p, \mathcal{T})$ of width k such that:*

1. *if $\mathcal{C}$ is closed under disjoint unions, then H contains no false twins, and*
2. *if $\mathcal{C}$ is closed under joins, then H contains no true twins.*

Proof. Towards a contradiction for (1), choose an H that contains the minimum number of false twins and suppose that this number is positive. Let $u, v \in V(H)$ be one such pair of false twins. The function p replaces each vertex of H with a graph from $\mathcal{C}$ to obtain G. As u, v are false twins in H it follows that $\{u, v\}$ is a module in H. Let us alter the decomposition by removing v to create H' and observe that G can still be created from H' by creating $p' = p$ except for setting $(p')^{-1}(u)$ to be $G[p^{-1}(u) \cup p^{-1}(v)]$. This subgraph is in $\mathcal{C}$ because both $p^{-1}(u)$ and $p^{-1}(v)$ are in $\mathcal{C}$ and in G there is no edge between them as $uv \notin E(H)$, and because $\mathcal{C}$ is closed under disjoint union. We removed v so the new decomposition $(H', p', \mathcal{T} - v)$ contains fewer twins, a contradiction.

We prove (2) by a similar contradiction; assuming $\mathcal{C}$ is closed under joins and that u, v are true twins in H, we show that v can be removed because $G[p^{-1}(u) \cup p^{-1}(v)]$ is by $uv \in E(H)$ a join of the two graphs from $\mathcal{C}$. ◀

▶ **Theorem 18.** *Given a graph G on n vertices there is an algorithm that in $n^{\mathcal{O}(1)}$ time retrieves a graph H and a mapping $p\colon V(G) \to V(H)$ such that for every $u \in V(H)$ the vertices $p^{-1}(u)$ form an independent set of G.*

Proof. We aim to find maximal modules that induce independent sets in the graph. $\mathcal{I}$ is closed under disjoint union so we invoke Lemma 17 and conclude that G has an $\mathcal{I}$-modular tree-decomposition $(H, p, \mathcal{T})$ where its module graph H contains no false twins. To find such H and p of such a decomposition we first retrieve the false twin relation by comparing open neighborhoods of all pairs of vertices; this implies a partition $\mathcal{P}$ of $V(G)$.

From each part $P \in \mathcal{P}$ we pick an arbitrary single representative $u^P \in P$. Let the modular graph H be the graph induced on the representatives $H = G[\{u^P \mid P \in \mathcal{P}\}]$. The above can be easily done in $n^{\mathcal{O}(1)}$ time. For every $v \in V(G)$ let $p(v) = u^P$ where $P \in \mathcal{P}$ such that $v \in P$. As $\mathcal{P}$ splits the graph into parts of equivalent neighborhoods, we have that the parts are modules of G. For any pair of vertices $v_1, v_2 \in V(G)$ we have $v_1 \sim_G v_2 \iff p(v_1) \sim_H p(v_2)$ by $N(p(v_1)) = N(v_1)$, $N(p(v_2)) = N(v_2)$, and H being an induced subgraph of G. ◀

To retrieve a cograph-modular tree-decomposition we use a very similar argument as for the independent-modular tree-decomposition; a number of notions needs to be introduced first.

Let us establish notations regarding partitions. Let each partition be represented as a family of sets that correspond to the parts, i.e, for a set $\{a, b, c, d, e, f\}$ we can have its partition $\{\{a, b, c\}, \{d, e\}, \{f\}\}$. The procedure described below creates nested partitions (partitions of partitions), resulting in e.g. $\{\{\{a, b, c\}\}, \{\{d, e\}, \{f\}\}\}$. Let *partition tree* $T(U)$ of a set U be a tree that corresponds to the bracket structure with labels of U replaced with $\emptyset$ – defined inductively, if U is a label let $T(U)$ be a rooted tree with only the root vertex, otherwise $U = \{u_1, \ldots, u_c\}$ and $T(U)$ is a rooted tree where the root has c children – roots of its subtrees $T(u_1), \ldots, T(u_c)$.

Let G be a graph with cograph-modular-treewidth k. Let $\mathcal{P}_c(G)$ be a partition of $V(G)$ such that u, v belong to the same part if and only if $N[u] = N[v]$. Let $\mathcal{Q}_c(G)$ be the quotient graph over $\mathcal{P}_c(G)$, i.e., graph $\mathcal{Q}_c(G) = (\mathcal{P}_c(G), \{\{P, Q\} \mid P, Q \in \mathcal{P}_c(G), P \neq Q, u \in P, v \in Q, uv \in E(G)\})$. Similarly for open neighborhoods, let $\mathcal{P}_o(G)$ be a partition where $u, v \in P \in \mathcal{P}_o(G)$ if and only if $N(u) = N(v)$ and let $\mathcal{Q}_o(G)$ be the quotient graph over $\mathcal{P}_o(G)$. Note that taking the quotient graph results in a graph that is isomorphic to a graph where all but one vertex is removed from each part because the considered parts are always modules of the graph.

▶ **Lemma 19.** *Suppose G' is a cograph with at least 2 vertices and G' is an induced subgraph of G and $V(G')$ is a module of G, then G' contains a pair of vertices that are true or false twins in G.*

Proof. As G' is a cograph it contains a pair of vertices that in $G[V(G')]$ are true or false twins [9, Theorem 2]. Suppose G' contains $u, v \in V(G')$ that are true twins in $G[V(G')]$, so $N[u] \cap V(G') = N[v] \cap V(G')$. As u, v are part of the module their neighborhood in $V(G) \setminus V(G')$ is identical. We conclude that u, v are true twins in G as well. If G' contains false twins instead, the proof is identical. ◀

Now, we use the above lemma to find a tree decomposition of the module graph by first identifying twins of G, alternating between identifying true or false twins, and then invoking the algorithm to find the tree decomposition of the resulting graph.

▶ **Theorem 20.** *Given a graph G on n vertices there is an algorithm that in $n^{\mathcal{O}(1)}$ time retrieves a graph H and a mapping $p\colon V(G) \to V(H)$ such that for every $u \in V(H)$ the induced subgraph $G[p^{-1}(u)]$ is a cograph.*

Proof. We define a sequence of graphs $G_0, G_1, \ldots, G_{2q}$ where $G_0 = G$ and for each $i \in [q]$ the graph $G_{2i-1} = \mathcal{Q}_c(G_{2i-2})$ and $G_{2i} = \mathcal{Q}_o(G_{2i-1})$; let q be the minimum such that $G_{2q} = \mathcal{Q}_c(\mathcal{Q}_o(G_{2q+2}))$. Note that $q \leq n$ as by the definition for each $i \in [q]$ the graphs G_{2i} and G_{2i-2} are distinct; by the construction we have $|V(G_{2i})| < |V(G_{2i-2})|$. I.e., in the above we constructed a sequence of quotient graphs over alternating partitions $\mathcal{P}_c$ and $\mathcal{P}_o$

which ends at a point where the application does not change the graph, i.e., all vertices have distinct open neighborhoods and distinct closed neighborhoods. All of this takes at most $n^{\mathcal{O}(1)}$ time as $q \leq n$, and each step takes at most polynomial-time.

Note that due to Lemma 17 graph H should not contain twins; the above process gets rid of them. Moreover, due to Lemma 19, if there is a module that induces a cograph then the process eventually reduces it to a single vertex. We set $H = G_{2q}$ which is a module graph of G and p can be constructed according to Lemma 17 in a straight-forward manner. ◀

To finish, we apply the algorithm of Korhonen [24] which either returns a tree-decomposition $\mathcal{T}$ of H of width $2k+1$ or concludes that the treewidth of H is more than k in $2^{\mathcal{O}(k)} \cdot |V(H)|$ time. We return $(H, p, \mathcal{T})$ as the independent-modular tree-decomposition.

▶ **Corollary 21.** *Given an integer k and a graph G on n vertices there is an algorithm that retrieves an independent-modular tree-decomposition of G of width $2k+1$ or determines that the independent-modular-treewidth of G is more than k in $n^{\mathcal{O}(1)} + 2^{\mathcal{O}(k)} \cdot n$ time.*

▶ **Corollary 22.** *Given an integer k and a graph G on n vertices there is an algorithm that retrieves a cograph-modular tree-decomposition of G of width $2k+1$ or determines that the cograph-modular-treewidth of G is more than k in $n^{\mathcal{O}(1)} + 2^{\mathcal{O}(k)} \cdot n$ time.*

Note that in the proof of Theorem 20 the labels assigned to vertices correspond to cotrees of the particular modules. This could be utilized to retrieve cotrees immediately, but for simplicity we will retrieve the cotrees of $p^{-1}(u)$ for each $u \in V(H)$ independently using [6].

4 Isomorphism/`cmtw` is FPT

In short, to tackle Isomorphism parameterized by `cmtw` we first retrieve the cograph-modular tree-decompositions. As every cograph is uniquely determined by its cotree (see Definition 10) we use the idea from the tree isomorphism checking algorithm to attach to each tree a unique number that represents its shape – two of these numbers are equal if and only if the two represented cotrees (and so their respective cographs) are isomorphic. Then, to each vertex v of the module graphs we attach a tree that represents the number for the cotree of $p^{-1}(v)$. These attached trees are shown not to occur in the module graph so we can safely use them to uniquely represent the cograph modules. In the end, we show that the attached trees are isomorphic if and only if the represented cographs are isomorphic. As attaching trees does not change treewidth of the graph it then suffices to check isomorphism of the two altered module graphs. Let us build these tools more formally now.

Let us first recall that isomorphism of rooted trees with unordered children.

▶ **Lemma 23** (see e.g. [1, Theorem 3.3])**.** *There is a procedure that assigns integers $\gamma\colon T \to [|T|]$ to each tree in a set of rooted trees T (with unordered children) so that two trees $t_1, t_2 \in T$ are isomorphic $t_1 \approx t_2$ if and only if their integers are equal $\gamma(t_1) = \gamma(t_2)$.*

Proof. The procedure assigns γ to every root of every subtree (i.e., every vertex of every tree) so that two subtrees are isomorphic if and only if their γ are equal. We can assume that the roots of all the trees in the set are children of a new dummy root so we can focus on processing a single tree. In essence, the vertices of the rooted tree are labelled bottom-up as follows. Each leaf gets label 0 and each inner vertex gets a label determined from the sorted labels of its children. More precisely, if label combination was never seen before it assigns the lowest unused $\mathbb{N}$; hence, the maximum integer used for a label is no more than the total number of vertices. It follows that isomorphic subtrees get the same label, and that the used

labels are upper bound by the number of processed vertices. To compare two different trees one needs to preserve the label assigning function from one tree and use it to label the other tree – two trees are deemed isomorphic if they end up having the same-labelled root. ◀

We will use Lemma 23 to get integers for cotrees of cographs that make up modules of our graph. These numbers will be used as labels and we solve the labeled instance using transformation from the following lemma.

▶ **Lemma 24.** *Given a pair of graphs G, G' and a labeling function $\delta\colon V(G)\cup V(G') \to [n^{\mathcal{O}(1)}]$ there is an algorithm that in $n^{\mathcal{O}(1)}$ time transforms graphs G, G' into graphs $\hat{G}, \hat{G}'$ so that $G \approx_\delta G'$, i.e., G has a δ-label-preserving isomorphism to G' if and only if $\hat{G} \approx \hat{G}'$, i.e., $\hat{G}$ is isomorphic to $\hat{G}'$.*

Proof. Let us create $\hat{G}$ from G by first subdiving all edges once and then to every vertex $u \in V(G)$ attaching $\delta(u) + 2$ new leaves. We create $\hat{G}'$ from G' in the same way. This transformation runs in $n^{\mathcal{O}(1)}$ time. Due to the subdivision the transformation creates a bipartite graph. All of the new leaves are in one part of the bipartition. On the other hand, the old leaves (those in G) are not leaves in $\hat{G}$ as to each of them we attached at least 2 new leaves. Note that the transformation is label agnostic and so if $G \approx_\delta G'$ then $\hat{G} \approx \hat{G}'$. In the other direction, assuming $\hat{G} \approx \hat{G}'$ we know that the parts with the leaves need to be mapped to each other. The number of adjacent leaves of a vertex is invariant under isomorphism and structure of the graph is clearly reflected in subdivision vertices, so it is straight-forward to lift the isomorphism to $G \approx_\delta G'$. ◀

▶ **Theorem 25.** *Given a pair of graphs G, G' and an integer k in $f(k) \cdot n^{\mathcal{O}(1)}$ time we can either decide* Isomorphism *of G and G' or conclude that either G or G' has cograph-modular-treewidth more than k.*

Proof. By Corollary 22 we either get cograph-modular tree-decomposition $(H, p, \mathcal{T})$ of G and $(H', p', \mathcal{T}')$ of G', each of width at most $2k + 1$, or conclude that one of the graphs has cograph-modular-treewidth more than k. Then, for each vertex $u \in V(H)$ we take the induced subgraph $G[p^{-1}(u)]$ and retrieve its cotree C_u; similarly we retrieve cotrees of all the vertices in G'. Now we create a labeling $\gamma\colon V(G) \cup V(G') \to \mathbb{N}$. Then, for every subgraph $\{G[p^{-1}(u)] \mid u \in V(H)\}$ we retrieve its cotree C_u [6]; similarly for H' we get cotrees C'_v. Now we run Lemma 23 for the set of all retrieved cotrees $\{C_u \mid u \in (V(G) \cup V(G'))\}$, let $\gamma(u) = \delta(C_u)$, i.e., we set $\gamma(u)$ to be the label that uniquely identifies the cotree C_u and subsequently also uniquely identifies the respective cograph. Note that γ has no values bigger than $|V(G)| + |V(G')|$. Next, we invoke Lemma 24 to transform H, H' to $\hat{H}, \hat{H}'$. Last, we use [26] to determine whether $\hat{H}$ is isomorphic to $\hat{H}'$ in $2^{\mathcal{O}(k^5 \log k)} \cdot (|V(\hat{H})| + |V(\hat{H}')|)^5$ time.

We claim that the answer to whether G is isomorphic to G' is equivalent to answering whether $\hat{H}$ is isomorphic to $\hat{H}'$. Indeed, observe that Corollary 22 obtains the cograph-modular tree-decomposition in a label-agnostic way so $G \approx G'$ if and only if the bijection that witnesses $H \approx H'$ maps to each other vertices that represent the same underlying cographs, i.e., a bijection $\alpha\colon H \to H'$ must satisfy $p^{-1}(u) \approx p'^{-1}(\alpha(u))$ for every $u \in V(H)$. We modeled this requirement by using bijection between cographs and cotrees, then using Lemma 23 to get the auxiliary labeling γ, and requiring γ-preserving isomorphism. Last, we used a transformation Lemma 24 which produces an equivalent instance. ◀

5 Chromatic Number/cmtw is W[1]-hard

A *coloring* of a graph G is an assignment $c\colon V(G) \to \mathbb{N}$ of a positive integer (color) to each vertex of G. The coloring c is *proper* if adjacent vertices receive distinct colors. The *chromatic number* $\chi(G)$ of a graph G is the smallest number of colors of a proper coloring of G. The Chromatic Number problem asks whether $\chi(G) \leq r$ of G. Note that Chromatic Number is not expressible in monadic second order logic. However, for every fixed r, checking whether $\chi(G) \leq r$ can be expressed in monadic second order logic even without edge set quantification. Since graphs of tree-width at most t are $t+1$ colorable, this implies that Chromatic Number can be solved in linear time on graph classes of bounded treewidth. Moreover, this problem admits FPT algorithm when parameterized by tw, although it becomes W[1]-hard when parameterized by cw [20]. In this section, we show that the Chromatic Number problem is W[1]-hard when the parameter is cograph-modular-treewidth. This hardness persists even when every cograph in the underlying decomposition is a clique. However, it admits an FPT algorithm when parameterized by independent-modular-treewidth [25]. In summary, we present the following result.

▶ **Theorem 26.** Chromatic Number/cmtw *is* W[1]*-hard.*

Proof. Our proof is based on a construction similar to that of [20], with specific modifications – namely, the addition of further gadget structures, which establishes the W[1]-hardness of the Chromatic Number when parameterized by clique-width. We give a reduction from Equitable Coloring/tw $+ r$ which is known to be W[1]-hard [18]. In the Equitable Coloring problem one is given a graph G on n vertices and integer r and asked whether G can be properly r-colored in such a way that the number of vertices in any two color classes differs by at most 1 (such coloring is called an *equitable r-coloring*). Notice that if n is divisible by r this implies that all color classes must contain the same number of vertices. In our reduction, we will assume that in the instance we reduce from, n is divisible by r. For a justification of this assumption, if r does not divide n we can add a clique of size $r - (n - \lfloor n/r \rfloor r)$ to G. We reduce from the exact version of Equitable Coloring, that is, the version where we are looking for an equitable coloring of G with exactly r colors.

Reduction. Given an input (G, r) to Equitable Coloring, we construct an instance (H, r') of Chromatic Number as follows (see Figure 3). Let $n = |V(G)|$ and $r' = r + nr$. We start with a copy G' of G. We add a clique Q of size nr and connect every vertex of Q to every vertex of G' by an edge. We add a clique P of size r' to H, called *palette*. The set P is partitioned into $r+1$ parts: $P = P^M \uplus P_1 \uplus P_2 \uplus \cdots \uplus P_r$, where $|P^M| = r$, $|P_i| = n$ (for $i \in [r]$). The vertices of P^M are denoted by $p_1, p_2, \ldots, p_r$. For each vertex $v \in P_M$ and each vertex $w \in Q$, we add the edge vw. For each $i \in [r]$, we add a set S_i of n vertices which contains copies of all vertices of G. For each vertex $u \in V(G)$ and each $i \in [r]$, we assign vertices $u_{P_i} \in P_i$ and $u_{S_i} \in S_i$. For each vertex $u \in V(G)$ and each $i \in [r]$, we add the edge uu_{P_i}. For every vertex $u \in V(G)$ and each $i \in [r]$, we make u_{S_i} adjacent to u and the entire palette P except for u_{P_i} and p_i. For each $i \in [r]$, we add a clique C_i of $\frac{n(r-1)}{r}$ vertices and make every vertex of C_i adjacent to all the vertices of S_i and the entire palette P except for P_i. Now we show the correctness of the reduction.

▷ Claim 27. If G has an equitable r-coloring ψ, then H has a proper r'-coloring φ.

▷ Claim 28. If H has a proper r'-coloring φ, then G has an equitable r-coloring ψ.

▷ Claim 29. $\texttt{cmtw}(H) \leq 2r \cdot \texttt{tw}(G) + r + 2$

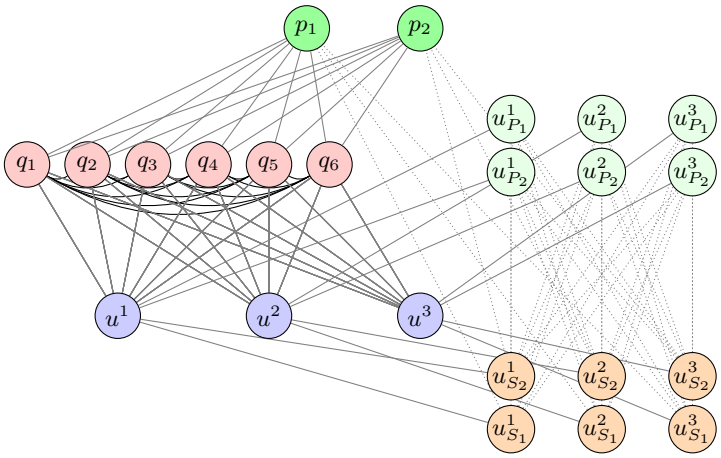


Figure 3 Illustration of the construction in Theorem 26 for $n = 3$, $r = 2$.

Claims 27, 28 and 29 together give a parameterized reduction from Equitable Coloring for $\mathtt{tw} + r$ into Chromatic Number/$\mathtt{cmtw}$. Moreover, as per our construction, this hardness holds even when every cograph in the underlying decomposition is a clique or an independent set. This completes the proof of Theorem 26. ◀

6 Edge Dominating Set/$\mathtt{imtw}$ is W[1]-hard

An *edge dominating set* of a graph G is a set $X \subseteq E(G)$ such that every edge of G is either in X or adjacent to at least one edge of X. Edge Dominating Set problem asks to find an edge dominating set of size at most k. This problem admits an FPT algorithm parameterized by $\mathtt{tw}$ by Courcelle's Theorem since it is MSO_2 expressible but it is W[1]-hard when parameterized by $\mathtt{cw}$ [20]. We show that this problem remains W[1]-hard when parameterized by $\mathtt{cmtw}$.

6.1 Exact Saturated Dominating Set

We first introduce a problem from which we will reduce. A *capacitated graph* is a pair (G, c), where G is a graph and $c\colon V(G) \to \mathbb{N}$ is a capacity function such that $c(v) \in [\deg(v)]$ for every vertex $v \in V(G)$ (sometimes we simply say that G is a capacitated graph if the capacity function is clear from the context). A set $S \subseteq V(G)$ is called a *capacitated dominating set* if there is a domination mapping $f\colon V(G) \setminus S \to S$ which maps every vertex in $V(G) \setminus S$ to one of its neighbors in S in such a way that the total number of vertices mapped by f to any vertex $v \in S$ does not exceed its capacity $c(v)$, i.e., $|f^{-1}(v)| \leq c(v)$. For a vertex $v \in S$ we say that vertices in $f^{-1}(v)$ are dominated by v. In Capacitated Dominating Set, given a capacitated graph (G, c) and a positive integer k, the objective is to check whether G has a capacitated dominating set of size at most k for G. In the Exact Capacitated Dominating Set problem, the size of such a set needs to be *exactly* k. It is known that both Capacitated Dominating Set and Exact Capacitated Dominating Set are W[1]-hard when parameterized by the treewidth of the input graph and the solution size k [17, 20].

For a capacitated graph (G, c), a capacitated dominating set $S \subseteq V(G)$ is called a *saturated dominating set* if there is a domination mapping f such that $|f^{-1}(v)| = c(v)$ for each vertex $v \in S$. Below we define the Exact Saturated Dominating Set problem.

Exact Saturated Dominating Set
Input: A capacitated graph (G, c), a positive integer k.
Question: Does G have a saturated dominating set with *exactly* k vertices?

It is known that Exact Saturated Dominating Set/`cw` is W[1]-hard [20]. We expand the aforementioned result and show that Exact Saturated Dominating Set/`tw` is W[1]-hard.

▶ **Theorem 30.** *The* Exact Saturated Dominating Set/`tw` *is* W[1]*-hard.*

6.2 Edge Dominating Set

In this section we show the following result.

▶ **Theorem 31.** Edge Dominating Set/`imtw` *is* W[1]*-hard.*

Proof. We give a reduction from Exact Saturated Dominating Set/`tw`. Our proof follows a construction similar to that in [20], with minor modifications, which establishes the W[1]-hardness of the Edge Dominating Set problem when parameterized by clique-width. We start with a description of an auxiliary gadget.

Auxiliary gadget. Let $s \leq t$ be positive integers. We construct a graph $F_{s,t}$ with the vertex set $\{x_1, \ldots, x_s, y_1, \ldots, y_s, z_1, \ldots, z_t\}$ and edges $x_i y_j$ for $1 \leq i, j \leq s$, and $y_i z_j$ for $1 \leq i \leq s,\ 1 \leq j \leq t$. Basically, we have a complete bipartite graph between the x_i's and the y_j's as well as between the y_i's and the z_j's. The vertices $z_1, \ldots, z_t$ are called the *roots* of $F_{s,t}$, see Figure 4.

Reduction. Let (G, c) be a capacitated graph with the vertex set $\{u_1, \ldots, u_n\}$, and k be a positive integer. We introduce three vertex sets $\{v_1, \ldots, v_n\}$, $\{v'_1, \ldots, v'_n\}$, and $\{w_1, \ldots, w_n\}$. Then for each $i \in [n]$, we add the edges (v_i, w_i) and (v_i, v'_i). Now for every vertex u_i, we introduce a set U_i with $c(u_i)$ vertices. For every edge $u_i u_j \in E(G)$, we connect all vertices of U_i with v_j and all vertices of U_j to v_i. We introduce vertex sets $\{a_1, \ldots, a_n\}$ and $\{b_1, \ldots, b_n\}$. Vertices a_i are made adjacent to all vertices of U_i, w_i, and b_i. For every vertex b_i we add a set R_i of $c(u_i) + 1$ vertices and make b_i adjacent to all the vertices of R_i. Then for each R_i, $i \in [n]$, we add vertex sets P_i and Q_i both of size $|R_i|$ and we make a complete bipartite graph between the vertices of P_i and Q_i as well as P_i and R_i. Let $P = P_1 \cup P_2 \cup \ldots \cup P_n$. We denote the obtained graph by G', see Figure 5. Let $r = \sum_{v \in V(G)} c(v)$. Finally, we introduce three copies of $F_{s,t}$: (i) a copy of $F_{n-k,n}$ with roots $\{a_1, \ldots, a_n\}$, (ii) a copy of $F_{k,n}$ with roots $\{b_1, \ldots, b_n\}$, and (iii) a copy of $F_{n,r+n}$ with roots in P. Let H be this final resulting graph.

▷ Claim 32. Any set of s edges incident with vertices $y_1, \ldots, y_s$ forms an edge dominating set in $F_{s,t}$. Furthermore, let G be a graph obtained by the union of $F_{s,t}$ with some other graph H such that $V(F_{s,t}) \cap V(H) = \{z_1, \ldots, z_t\}$. Then every edge dominating set of G contains at least s edges from $F_{s,t}$.

The proof of Claim 32 follows from the fact that every edge dominating set includes at least one edge from $E(y_i)$ for $i \in \{1, \ldots, s\}$ where $E(y_i)$ denotes the set of edges with one endpoint at y_i.

▷ Claim 33. If the capacitated graph (G, c) has an exact saturated dominating set of size k then H has an edge dominating set of cardinality $3n + r$.

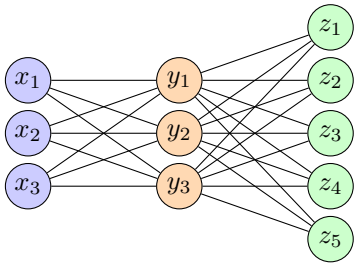


Figure 4 The graph $F_{3,5}$.

▷ Claim 34. If H has an edge dominating set of cardinality at most $3n + r$ then (G, c) has an exact saturated dominating set of size k.

▷ Claim 35. $\texttt{imtw}(H) \leq 7\texttt{tw}(G) + 6$

Claims 33, 34 and 35 together give a parameterized reduction from Exact Saturated Dominating Set/tw to Edge Dominating Set/imtw. This completes the proof of Theorem 31. ◀

7 Hamiltonian Cycle/imtw is W[1]-hard

A *Hamiltonian cycle* in a graph G is defined as a cycle that includes every vertex of G. The Hamiltonian Cycle problem asks finding such a cycle in G. Although the problem can be solved using an FPT algorithm with the parameterization by tw (which is expressible in MSO_2), it becomes W[1]-hard when parameterized by cw [20]. We demonstrate that the problem also remains W[1]-hard with the cmtw parameterization, and even W[1]-hard when using imtw as a parameter. In this section, we present the following finding.

▶ **Theorem 36.** Hamiltonian Cycle/imtw *is* W[1]*-hard.*

Proof. In the proof of W[1]-hardness of Hamiltonian Cycle/cw by Fomin et al. [20], the constructed gadget contains large independent modules. These modules have the property that, if each were contracted into a single vertex, the resulting graph would have bounded treewidth. We exhibit this property towards proving Theorem 36. For the sake of completeness, we present the same construction below, omitting the proof of the reduction (as it remains identical), but we include a proof demonstrating that the underlying graph has bounded treewidth.

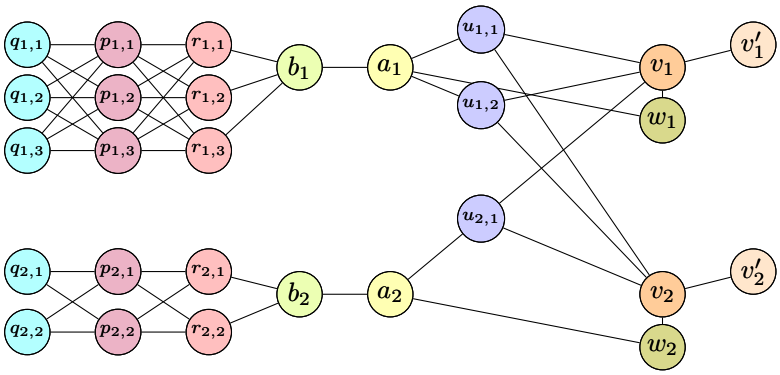


Figure 5 Illustration of the construction in Theorem 31 with $n = 2$, $c(u_1) = 2$, $c(u_2) = 1$.

We give a reduction from Capacitated Dominating Set/tw which is known to be W[1]-hard [20]. We start with descriptions of auxiliary gadgets.

First auxiliary gadget L_1. The first auxiliary gadget is the graph L_1 with the vertex set $\{x, y, z, a, b, c, d\}$ and the edge set $\{xa, ab, bc, cd, dy, bz, cz\}$. Let P_1 be the path $xabzcdy$ and $P_2 = xabcdy$.

Second auxiliary gadget L_2. The second auxiliary gadget is the graph L_2 with the vertex set $\{x, y, z, s, t, a, b, c, d, e, f, g, h\}$. Initially, the edges $\{xa, ab, bz, cz, cd, dy, se, ef, fb, ch, hg, gt\}$ in are added to its edge set. Then an x-y path $xw_1 \cdots w_9y$ of length 10 is added, and edges fw_3, w_1w_6, w_4w_9, w_7h are included in the set of edges. Let $P = xabzcdy$, $R_1 = sefbaxw_1w_2 \dots w_9ydchgt$, and $R_2 = sefw_3w_2w_1w_6w_5w_4w_9w_8w_7hgt$.

Reduction. Let (G, c) be a capacitated graph with the vertex set $\{v_1, \dots, v_n\}$ and m edges, and let k be a positive integer. The construction of H is as follows:

- For every vertex v_i, four vertices a_i, b_i, c_i, and w_i are introduced, and the vertices b_i and c_i are joined by $c(v_i) + 1$ paths of length two. Let C_i denote the set of middle vertices of these paths, and $X_i = C_i \cup \{a_i, b_i, c_i\}$. Then a copy L_i^2 of the graph L_2 with $z = w_i$ is added, and vertices x and y of this gadget are joined by edges to a_i and b_i, respectively. By s_i and t_i we denote the vertices s and t, respectively, of L_i^2. The paths corresponding to P in L_i^2 is called P^i.
- For every edge $v_iv_j \in E(G)$ with $i < j$, a copy L_{ij}^2 of L_2 is attached with $z = w_j$ and vertices x and y made adjacent to all the vertices of C_i. The vertices corresponding to s and t are called s_{ij} and t_{ij}, respectively, in L_{ij}^2. Furthermore, let x_{ij} and y_{ij} denote the vertices corresponding to x and y, respectively, in L_{ij}^2. The paths corresponding to P, R_1, and R_2 are called P^{ij}, R_1^{ij}, and R_2^{ij}, respectively, in L_{ij}^2.
- Next we add two vertices g and h which are joined by $\sum_{i=1}^{n}(c(v_i) + 4) + n + 2m + 1$ paths of length two. Let Y be the set of middle vertices of these paths. All vertices s_i, t_i, s_{ij}, and t_{ij} are joined by edges with all vertices of Y. For every vertex $r \in X_i$, $i \in [n]$, a copy L_r^1 of L_1 with $z = r$ is attached and the vertices x and y of this gadget are joined to all vertices of Y. We let x_r and y_r denote the vertices corresponding to x and y, respectively, in L_r^1. Similarly, P_1^r and P_2^r denote paths in L_r^1 corresponding to P_1 and P_2, respectively.
- Finally, we add $k + 1$ vertices, namely $\{p_1, \dots, p_{k+1}\}$, and make them adjacent to all the vertices $\{a_i, c_i \mid 1 \leq i \leq n\}$ and to g and h.

▷ Claim 37 ([20, Lemma 10]). (G, c) has a capacitated dominating set of size at most k if and only if H has a Hamiltonian cycle.

▷ Claim 38. $\texttt{imtw}(H) \leq 24\texttt{tw}(G)$

Claims 37 and 38 together give a parameterized reduction from Capacitated Dominating Set/tw to Edge Dominating Set/imtw. This completes the proof of Theorem 36. ◀

8 Conclusion

We introduced $\mathcal{C}$-modular-treewidth, a new graph parameter generalizing classical treewidth by leveraging structure from a fixed graph class $\mathcal{C}$. We established the parameterized complexity of several problems: both Hamiltonian Cycle and Edge Dominating Set are W[1]-hard for independent-modular-treewidth, while Graph Isomorphism is FPT for cograph-modular-treewidth. We also showed that Chromatic Number is W[1]-hard for cograph-modular-treewidth. Future work includes studying these and related problems under clique-modular-treewidth and exploring how different choices of the base class $\mathcal{C}$ affect the algorithmic tractability of classic problems.

References

1 Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1st edition, 1974.

2 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.

3 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.

4 Hans L. Bodlaender and Klaus Jansen. On the complexity of the maximum cut problem. *Nord. J. Comput.*, 7(1):14–31, March 2000.

5 Hans L. Bodlaender and Arie M.C.A. Koster. Combinatorial optimization on graphs of bounded treewidth. *Comput. J.*, 51(3):255–269, 2008. doi:10.1093/COMJNL/BXM037.

6 Anna Bretscher, Derek Corneil, Michel Habib, and Christophe Paul. A simple linear time LexBFS cograph recognition algorithm. *SIAM Journal on Discrete Mathematics*, 22(4):1277–1296, 2008. doi:10.1137/060664690.

7 Derek G. Corneil, Michel Habib, Christophe Paul, and Marc Tedder. A recursive linear time modular decomposition algorithm via LexBFS, 2024. arXiv:0710.3901.

8 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.

9 D.G. Corneil, H. Lerchs, and L. Stewart Burlingham. Complement reducible graphs. *Discrete Applied Mathematics*, 3(3):163–174, 1981. doi:10.1016/0166-218X(81)90013-5.

10 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.

11 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.

12 Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000. doi:10.1007/S002249910009.

13 Bruno Courcelle and Mohamed Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theor. Comput. Sci.*, 109(1&2):49–82, 1993. doi:10.1016/0304-3975(93)90064-Z.

14 Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1):77–114, 2000. doi:10.1016/S0166-218X(99)00184-5.

15 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 5. Springer, 2015. doi:10.1007/978-3-319-21275-3.

16 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. *ACM Trans. Algorithms*, 18(2):17:1–17:31, 2022. doi:10.1145/3506707.

17 Michael Dom, Daniel Lokshtanov, Saket Saurabh, and Yngve Villanger. Capacitated domination and covering: A parameterized perspective. In *Parameterized and Exact Computation, Third International Workshop, IWPEC*, volume 5018 of *Lecture Notes in Computer Science*, pages 78–90. Springer, 2008. doi:10.1007/978-3-540-79723-4_9.

18 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.

19 Jacob Focke, Dániel Marx, and Pawel Rzazewski. Counting list homomorphisms from graphs of bounded treewidth: Tight complexity bounds. *ACM Trans. Algorithms*, 20(2):11, 2024. doi:10.1145/3640814.

20 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Intractability of clique-width parameterizations. *SIAM J. Comput.*, 39(5):1941–1956, 2010. doi:10.1137/080742270.

21 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Almost optimal lower bounds for problems parameterized by clique-width. *SIAM J. Comput.*, 43(5):1541–1563, 2014. doi:10.1137/130910932.

22 Martin Grohe and Pascal Schweitzer. Isomorphism testing for graphs of bounded rank width. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1010–1029. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.66.

23 Falko Hegerfeld and Stefan Kratsch. Tight algorithms for connectivity problems parameterized by modular-treewidth. In Daniël Paulusma and Bernard Ries, editors, *Graph-Theoretic Concepts in Computer Science - 49th International Workshop, WG 2023, Fribourg, Switzerland, June 28-30, 2023, Revised Selected Papers*, volume 14093 of *Lecture Notes in Computer Science*, pages 388–402. Springer, 2023. doi:10.1007/978-3-031-43380-1_28.

24 Tuukka Korhonen. A single-exponential time 2-approximation algorithm for treewidth. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 184–192, 2022. doi:10.1109/FOCS52979.2021.00026.

25 Michael Lampis. Finer tight bounds for coloring on clique-width. *SIAM Journal on Discrete Mathematics*, 34(3):1538–1558, 2020. doi:10.1137/19M1280326.

26 Daniel Lokshtanov, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. Fixed-parameter tractable canonization and isomorphism test for graphs of bounded treewidth. *SIAM J. Comput.*, 46(1):161–189, 2017. doi:10.1137/140999980.

27 Stefan Mengel. Parameterized compilation lower bounds for restricted cnf-formulas. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, volume 9710 of *Lecture Notes in Computer Science*, pages 3–12. Springer, 2016. doi:10.1007/978-3-319-40970-2_1.

28 Karolina Okrasa, Marta Piecyk, and Pawel Rzazewski. Full complexity classification of the list homomorphism problem for bounded-treewidth graphs. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms, ESA*, volume 173 of *LIPIcs*, pages 74:1–74:24. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.ESA.2020.74.

29 Daniël Paulusma, Friedrich Slivovsky, and Stefan Szeider. Model counting for CNF formulas of bounded modular treewidth. *Algorithmica*, 76(1):168–194, 2016. doi:10.1007/S00453-015-0030-X.

30 Sigve Hortemo Sæther and Jan Arne Telle. Between treewidth and clique-width. *Algorithmica*, 75(1):218–253, 2016. doi:10.1007/S00453-015-0033-7.

31 Marc Tedder, Derek G. Corneil, Michel Habib, and Christophe Paul. Simpler linear-time modular decomposition via recursive factorizing permutations. In *Automata, Languages and Programming*, pages 634–645, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. doi:10.1007/978-3-540-70575-8_52.

32 Johan M. M. van Rooij, Hans L. Bodlaender, and Peter Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In *Algorithms – ESA 2009*, volume 5757 of *Lecture Notes in Computer Science*, pages 566–577. Springer, 2009. doi:10.1007/978-3-642-04128-0_51.

Tight Bounds for Connected Odd Cycle Transversal Parameterized by Clique-Width

Narek Bojikian
Humboldt-Universität zu Berlin, Germany

Stefan Kratsch
Humboldt-Universität zu Berlin, Germany

Abstract

Recently, Bojikian and Kratsch [ICALP 2024] presented a novel approach to tackle connectivity problems parameterized by clique-width (cw), based on counting (modulo 2) the number of representations of partial solutions, while allowing for possibly multiple representations to exist for the same partial solution. Using this technique, they got a SETH-tight bound of $\mathcal{O}^*(3^{\text{cw}})$ for the Steiner Tree problem, which was left open by Hegerfeld and Kratsch [ESA 2023]. We use the same technique to solve the Connected Odd Cycle Transversal problem in time $\mathcal{O}^*(12^{\text{cw}})$. Moreover, we prove that our result is tight by providing a SETH-based lower bound excluding algorithms with running time $\mathcal{O}^*((12-\epsilon)^{\text{cw}})$. This answers another question of Hegerfeld and Kratsch [ESA 2023].

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

Keywords and phrases Parameterized complexity, connected odd cycle transversal, clique-width

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.19

Related Version *Full Version*: `https://arxiv.org/abs/2402.08046` [7]

Acknowledgements We are grateful to Vera Chekan for her detailed comments on presentation, and to Falko Hegerfeld for several discussions on the lower bound presented here.

1 Introduction

Parameterized complexity studies the fine-grained complexity of problems by analyzing the dependence of the running time on different measures of the input (called parameters) in addition to its size. Typical parameters are the solution size or structural measures relating, e.g., to sparsity or the existence of certain decompositions of the input. We refer to [12] for an introduction to the field. A central goal is to determine whether a problem with chosen parameter t can be solved in time $f(t) \cdot n^{\mathcal{O}(1)}$ where f is an arbitrary computable function and n is the input size. Such problems are called *fixed-parameter tractable* and form the class FPT. While in general the dependence on t may be arbitrary, there is much interest in making this as low as possible and possibly establishing conditional optimality of obtained time bounds.

The most studied parameter in this direction is the treewidth (of a graph), a measure that sparks interest in different areas of theoretical computer science, not only parameterized complexity. Roughly, the treewidth of a graph G is the smallest value k such that G can be fully decomposed along non-crossing separators of size k. FPT algorithms with single exponential dependence on the treewidth of the graph have long been known for many problems. However, one would still try to improve the exponential dependence on the

Due to space constraints, we defer full proofs and technical details to the full version [7].

© Narek Bojikian and Stefan Kratsch;
licensed under Creative Commons License CC-BY 4.0

parameter, by improving the base of the exponent in the running time. In 2010, Lokshtanov et al. [33, 34] provided the first tight lower bounds for problems parameterized by treewidth, proving that known bases for some benchmark problems cannot be improved assuming the strong exponential time hypothesis (SETH), a widely used hypothesis that, informally speaking, conjectures that the naive solution of the SAT problem by testing all possible assignments is essentially optimal [27, 28]. This has inspired a series of SETH based lower bounds for structural parameters [11, 16, 36, 32, 22, 24].

Connectivity problems like Steiner Tree or Feedback Vertex Set, however, resisted single exponential running time algorithms for a long time. A major obstacle had been keeping track of different connectivity patterns in the boundary of the graph, representing different partial solutions in this graph. This seemed to necessitate a factor of $k^{\mathcal{O}(k)}$ in the running time. In 2011, Cygan et al. [14, 15] showed that this issue can be overcome for many (but not all) connectivity problems, by means of their Cut&Count technique. The core idea is to count the number of connected solutions of a problem modulo 2 in single exponential time by counting the ways that (possibly disconnected) solution candidates can be cut. They used the isolation lemma [35] to solve the underlying decision problem with high probability. Finally, they also proved that many of the resulting bases were tight under SETH.

Relying on the Cut&Count technique, tight bounds for connectivity problems were also derived relative to parameters such as pathwidth [13], cutwidth [6], modular treewidth [25], and clique-width [24, 8]. In this paper, we focus on Clique-width, a graph parameter defined by the smallest value k, such that a given graph can be built recursively from the disjoint union of k-labeled graphs, by allowing to add bicliques between the vertices of two label classes, and to relabel all vertices in a class. Clique-width is a more general parameter than treewidth, since cliques have clique-width at most 2, and unbounded treewidth, while graphs of treewidth k have bounded clique-width (at most $2^{\mathcal{O}(k)}$) [10, 9].

The first single exponential time algorithms for connectivity problems parameterized by clique-width, to the best of our knowledge, were given by Bergougnoux and Kanté [2]. However, the first SETH-tight bounds were given by Hegerfeld and Kratsch [24], where they provided algorithms, based on the Cut&Count technique, and matching SETH-based lower bounds for the Connected Vertex Cover and the Connected Dominating Set problems. They left open however, the tight complexity of other connectivity problems such as Steiner Tree and the Connected Odd Cycle Transversal. A major obstacle towards finding the optimal base of the exponent, as they mention, is a gap between the rank of a matrix defined by the interactions of partial solutions of the underlying problem (called *the compatibility matrix*), and its largest triangular submatrix. Except for a handful examples (Feedback Vertex Set [ctw] [6]), the rank of such matrices has usually been used directly to derive the optimal running time, while current lower bound techniques usually match the size of a largest triangular submatrix.

Very recently Bojikian and Kratsch [8] introduced a new technique called "isolating a representative" closing the gap for the Steiner Tree problem. Their technique is based on a smaller set of possible representations of connectivity patterns. They showed that one can count the number of these representations (modulo 2) in time proportional to the size of a largest triangular submatrix of the corresponding compatibility matrix. They introduced the notion of action-sequences to distinguish different representations of the same solution, and using the isolation lemma to also isolate a unique representation, they were able to solve the underlying decision problem in the same running time, matching known lower bounds.

Our Technique. Following the approach of Bojikian and Kratsch, we provide an optimal (modulo SETH) algorithm for the Connected Odd Cycle Transversal problem parameterized by clique-width with running time $\mathcal{O}^*(12^k)$, proving the following theorem (we refer to Section 2 for definitions):

▶ **Theorem 1.** *There exists a one-sided error Monte-Carlo algorithm (with false negatives only), that given a graph G together with a k-expression of G, and a positive integer $\overline{b}$, solves the* Connected Odd Cycle Transversal *problem in time $\mathcal{O}^*(12^k)$, and outputs the correct answer with probability at least $1/2$.*

Our algorithm counts the number of representations of solutions modulo 2. We achieve this by double representation of partial solutions. The first of which is existential, in the sense that we replace each family of partial solutions at a node of the clique-expression with a smaller family, such that the former completes into a solution in the full graph if and only if the representing family does. In the second step we define an even smaller family, that correctly counts (modulo 2) the number of extensions of the first family into solutions in the rest of the graph. Finally, we assign weights to different actions defined to build the first representing family, and we use the isolation lemma [35] to isolate a single representation with high probability. We make use of fast convolution methods over power lattices [4, 24, 6].

We prove the tightness of our algorithm under SETH by a parameter-preserving reduction from the d-SAT problem. To the best of our knowledge, this is the largest known single exponential base for a natural problem parameterized by a structural parameter. This imposes a challenge when designing the lower bound. As is usual, the lower bound even works for the linear version of the parameter, i.e., for *linear clique-width* [1, 20, 26], and we obtain the following theorem:

▶ **Theorem 2.** *Given a graph G of linear clique-width at most k, assuming SETH, there exists no algorithm that solves the* Connected Odd Cycle Transversal *problem in time $\mathcal{O}^*\big((12-\varepsilon)^k\big)$ even when a linear k-expression of G is provided with G.*

Related work. Clique-width was first studied by Courcelle and Olariu [10] showing that problems expressible in MSO_1 can be solved in FPT time when parameterized by clique-width. However, the resulting algorithms could have a non-elementary dependence on the parameter value k, and hence, are probably not tight for most relevant problems. Espelage et al. [17] provided XP algorithms for some problems not expressible in MSO_1 logic. Nonetheless, some problems were proven to be para-NP-hard when parameterized by clique-width, and hence probably do not admit a polynomial time algorithm on graphs of bounded clique-width [21].

The SETH-tight lower bounds by Lokshtanov et al. [34] were followed by series of tight bounds for structural parameters [6, 36, 19, 30, 32]. Iwata and Yoshida [29] proved that Vertex Cover has the same base parameterized by treewidth and clique-width, providing one of the earliest tight bounds for clique-width. Further tight bounds were obtained for counting perfect matchings, graph coloring and other problems [11, 18, 24, 31, 32].

Finally, we make use of fast convolution methods. These methods are usually used to combine different partial solutions in different parts of the graph along a common boundary set efficiently. Björklund et al. [3] introduced fast convolution over the subset lattice solving the Steiner Tree problem parameterized by the number of terminals more efficiently. Fast subset convolution, and variations thereof have since become a standard technique to handle a join node in dynamic programming algorithms over tree decompositions [37, 15]. Björklund et al. [4] showed that one can compute the so-called join-product over a lattice more efficiently,

if the underlying lattice admits a small number of so-called irreducible elements. Building on this result, Hegerfeld and Kratsch [24] provided a fast convolution technique over power lattices resulting in tight algorithms for connectivity problems parameterized by clique-width.

Organization. In Section 2 we provide preliminaries and introduce some notation. In Section 3 we define connectivity patterns and pattern operations. In Section 4 we describe the dynamic programming algorithm and prove its running time. In Section 5 we define action-sequences, and we use them to prove the correctness of the algorithm. Finally, in Section 6 we present the lower bound. We conclude with final remarks in Section 7.

2 Preliminaries

We denote by $[k]$ the set $\{1, 2, \dots k\}$ and by $[k]_0$ the set $[k] \cup \{0\}$. A *labeled graph* is a graph $G = (V, E)$ together with a *labeling function* $\text{lab}\colon V \to \mathbb{N}$. We usually omit lab and assume that it is given implicitly with G. We define a *clique-expression* μ as a well-formed expression defined by the following operations on labeled graphs:

- *Introduce* vertex $i(v)$ for $i \in \mathbb{N}$: constructs a graph with a single vertex labeled i.
- *Union* $G_1 \uplus G_2$: the disjoint union of G_1 and G_2.
- *Relabel* $\rho_{i\to j}(G)$ for $i \neq j \in \mathbb{N}$: change the label of all vertices labeled i to j.
- *Join* $\eta_{i,j}(G)$ for $i \neq j \in \mathbb{N}$: add all edges between vertices labeled i and vertices labeled j.

We denote the graph resulting from a clique-expression μ by G_μ, and the constructed labeling function by lab_μ. We associate with a clique-expression μ a syntax tree $\mathcal{T}_\mu$ in the natural way, and associate with each node $x \in V(\mathcal{T})$ the corresponding operation. Let $\mathcal{V}_\mu = V(\mathcal{T}_\mu)$. We omit μ when clear from the context. The subtree rooted at each node x induces a subexpression μ_x. We define $G_x = G_{\mu_x}$, $V_x = V(G_x)$, $E_x = E(G_x)$, $\text{lab}_x = \text{lab}_{\mu_x}$, $\mathcal{T}_x = \mathcal{T}_{\mu_x}$, and $\mathcal{V}_x = \mathcal{V}_{\mu_x}$. We also define the set $\mathcal{V}_x^C$ as the set of all *introduce* nodes in $\mathcal{T}_x$, and $\mathcal{V}_x^{CJ}$ as the set of all *introduce and join* nodes in $\mathcal{T}_x$. Given a set $S \subseteq V$, we define $S_x = S \cap V_x$. For a mapping $g\colon S \to U$, we denote by g_x the mapping $g_{|S_x}$, i.e. the restriction of g to $S \cap V_x$.

We call a clique-expression μ *linear*, if it holds for each union node x with children x_1 and x_2 that μ_{x_2} is an introduce operation. We say that G is k-labeled, if it holds that $\text{lab}(v) \leq k$ for all $v \in V(G)$. We say that a clique-expression μ is a *k-expression* if G_x is a k-labeled graph for all $x \in \mathcal{V}$. We define the (linear) clique-width of a graph G, denoted by $\text{cw}(G)$ ($\text{lcw}(G)$ resp.) as the smallest value k such that there exists a (linear) k-expression μ, with G_μ isomorphic to G. We can assume without loss of generality, that any given k-expression defining a graph $G = (V, E)$ uses at most $O(|V|)$ union operations, and at most $O(|V|k^2)$ unary operations [2, 10].

A mapping $g : V \to [q]$, for $q \in \mathbb{N}$, is a *proper q-coloring* of G, iff for all $\{u, v\} \in E$ it holds that $g(u) \neq g(v)$. A graph G is *q-colorable* if it admits a proper q-coloring. A set $S \subseteq V$ is an *odd cycle transversal* of G, if $G \setminus S$ is 2-colorable. In this case, we call a proper 2-coloring g of $G \setminus S$ a *witness*. In the Connected Odd Cycle Transversal problem, given is a graph G and a positive integer $\bar{b}$, asked is whether G admits an odd cycle transversal S of size at most $\bar{b}$, such that S induces a connected subgraph of G.

We denote the symmetric difference by Δ. A *weight function* ω is a mapping from a set U to some ring (usually $\mathbb{Z}$). Given $S \subseteq U$, we define $\omega(S) = \sum_{u\in S} \omega(u)$. Let $f : U^k \to U$ be a mapping (not explicitly defined as a weight function). For $S_1, \dots S_k \subseteq U$, we define $f(S_1, \dots S_k) = \{f(u_1, \dots, u_k)\colon u_i \in S_i\}$. For $S \subseteq U$, we define $f^{-1}(S)$ in a natural way. Finally, we define the operation $\overset{\triangle}{f}\colon (2^U)^k \to 2^U$ with $\overset{\triangle}{f}(S_1, \dots, S_k) = \triangle_{(u_1,\dots,u_k)\in S_1\times\cdots\times S_k} \big\{f(u_1, \dots, u_k)\big\}$, and call it *exclusive* f. We define the union of two mappings $f \dot\cup g$ over disjoint domains in a natural way.

A short introduction to lattices can be found in the full version. For a lattice $(\mathcal{L}, \leq)$, we define the set of *join-irreducible* elements $\mathcal{L}_\vee \subseteq \mathcal{L}$ as the set of elements $u \in \mathcal{L}$ where for all $v, w \in \mathcal{L}$ with $u = v \vee w$ it holds that $u = v$ or $u = w$. We say that a lattice $(\mathcal{L}, \leq)$ is given in the *join representation* if its elements are given as $\mathcal{O}(\log |\mathcal{L}|)$-bit strings, together with the elements of $\mathcal{L}_\vee$, and an algorithm $\mathcal{A}_\mathcal{L}$ that computes the join $u \vee v$ for $u \in \mathcal{L}$ and $v \in \mathcal{L}_\vee$.

Let $A, B \in \mathbb{F}^\mathcal{L}$ be two vectors over a ring $\mathbb{F}$. We define the $\vee$-*product* $A \otimes B \in \mathbb{F}^\mathcal{L}$ where for $z \in \mathcal{L}$ it holds that $A \otimes B[z] = \sum_{\substack{x,y \in \mathcal{L} \\ x \vee y = z}} A[x] \cdot B[y]$. We refer to the full version for details.

Conventions. Along the upper bound, let $G = (V, E)$ be the input graph and let $\overline{b} \in \mathbb{N}$ be the target value in the Connected Odd Cycle Transversal instance. Let v_0 be a fixed vertex that we choose later. Let $n = |V|$ and $m = |E|$. Let μ a k-expression of G for some value $k \in \mathbb{N}$, and $\mathcal{T}$ be the corresponding syntax tree. Let r be the root of $\mathcal{T}$. A *partial solution* at $x \in \mathcal{V}$ is a set of vertices $S \subseteq V_x$, representing the part of the solution introduced so far, such that $G_x \setminus S$ is bipartite.

For a function f, we denote by $\mathcal{O}^*(f(k))$ the class $f(k) \cdot n^{\mathcal{O}(1)}$, where the star hides any polynomial factor of n. We define the ground set $U = [k]$, and call its elements *labels*. Let $U_0 = [k]_0$. Let $W \in \mathbb{N}$ be some fixed value, and $\omega : \mathcal{V} \times [4] \to [W]$ a weight function, both fixed later. We define the intervals $\overline{B} = [n]_0$ and $\overline{W} = [|\mathcal{V}| \cdot W]_0$, and we use the indices b and w to iterate over $\overline{B}$ and $\overline{W}$ respectively. We skip defining these indices repeatedly to avoid redundancy.

3 Connectivity patterns

Patterns are structures that represent connectivity in labeled graphs. We use the definition of patterns and some of their properties from [8]. A *pattern* p is a set of subsets of U_0, such that there exists exactly one set $Z_p \in p$ that contains the element 0 in it. We call this set the *zero-set* of p. We denote by $\mathscr{P}$ the family of all patterns. Let $\text{label}(p) \subseteq U$ be set of labels that appear in at least one set of p, and $\text{singleton}(p) \subseteq U$ the set of labels that appear as singletons in p (excluding the element 0). We define the set of *incomplete labels* $\text{inc}(p) = \text{label}(p) \setminus \text{singleton}(p)$. We denote a pattern $\{\{i_1^1, \ldots, i_{k_1}^1\}, \ldots, \{i_1^\ell, \ldots, i_{k_\ell}^\ell\}\}$ by $\left[i_1^1 \ldots i_{k_1}^1, \cdots, i_1^\ell \ldots i_{k_\ell}^\ell\right]$, but only if this notation does not cause confusion. We also omit the zero-set when it is a singleton.

▶ **Definition 3.** *Given a labeled graph G and a set $S \subseteq V(G)$, we define the pattern* $\text{pat}_G(S)$ *as the sets of labels appearing in each connected component of $G[S]$. Formally, for each connected component C, we add the set* $\text{lab}(C)$ *to the pattern. We add the label* 0 *to the set corresponding to the component that contains v_0, or as singleton if $v_0 \notin S$.*

Such patterns carry enough information about the connectivity of partial solutions. We define pattern operations, that allow us to build connectivity patterns corresponding to different partial solutions recursively over the nodes of $\mathcal{T}$ (see [8] for details).

▶ **Definition 4.** *We define the operations* join, relabel, union, *and* add *over patterns as follows: The union operation of two patterns $p \oplus q$ is defined as the pattern r resulting from the union of p and q by replacing the zero-sets of both patterns by the union of these two sets. The relabel operation ($p_{i \to j}$) changes all occurrences of the label i into the label j. The join operation ($p \sqcup q$) exhaustively merges all sets of p and q that intersect. Finally, we define the add operation $\square_{i,j}(p)$ as $p \sqcup [ij]$ if $\{i, j\} \subseteq \text{label}(p)$, or as p otherwise.*

As we shall see later, these operations allow to build all patterns corresponding to all partial solutions of a fixed size over $\mathcal{T}$. This can be done by deciding at each leaf node, whether to include the introduced vertex in the solution or not. One can then compute these families at each node by applying the corresponding pattern operations on the families at the children nodes of this node. It is not hard to see that a union node corresponds to a union operation, a relabel node corresponds to a relabel operation, and a join node to an add operation. A correctness proof of this is given later implicitly, when proving that the so-called solution-sequences correspond to all partial solutions of a given weight.

Now we define the compatibility relation over patterns. This relation indicates whether two partial solutions in two labeled subgraphs join into a connected graph by connecting all vertices of the same label between them. Formally, we call two patterns p and q *compatible* ($p \sim q$), if their join is a single set.

This notion of compatibility will allow us to define two kinds of representation, an existential one (representation), and a (modulo 2) count-preserving one (parity-representation). Based on these, we identify two special families of patterns, namely the family of *complete patterns* $\mathscr{P}_C$ and the family of *CS-patterns* $\mathscr{P}_{CS}$. We will show later that one can represent each family of patterns by a family of complete patterns, and parity-represent each family of complete patterns by a family of CS-patterns.

▶ **Definition 5.** *A pattern p is* complete *if it holds that* $\text{singleton}(p) = \text{label}(p)$*, i.e. each label that appears in this pattern, appears as a singleton as well. We call a complete pattern a* CS-pattern*, if it consists only of a zero-set and singletons. We denote the family of* complete patterns *by $\mathscr{P}_C$ and the family of* CS-patterns *by $\mathscr{P}_{CS}$.*

▶ **Lemma 6.** *The pattern* $[0]$ *is the only complete pattern compatible with the pattern* $[0]$*.*

We will show that all defined pattern operations preserve (parity-)representation, in the sense that if R represents S, then $\text{Op}(R)$ represents $\text{Op}(S)$ as well. Therefore, we can replace the families of patterns of all partial solutions over $\mathcal{T}$ by representations thereof, which allows us to restrict the dynamic programming algorithm to CS-patterns only, correctly counting (modulo 2) the number of all representations of all solutions. Therefore, the running time will depend on the number of CS-patterns over k labels instead of the number of all patterns.

Finally, we use the isolation lemma to show that, by assigning random weights to the different representations of different partial solutions, we get a unique minimum-weight representation with high probability. This allows to solve the decision problem in the given time. The running time is implied by the number of CS-patterns over k labels, combined with a proper 2-coloring of the remaining part of the graph.

Before we present the algorithm, we still need to define the notion of *actions* and the operation PREP. Actions are series of "forget" and "fix" operations over patterns, that build a representation of a pattern by deciding whether a given label will still be relevant (in the future) for deciding the connectivity of a solution. Using actions, we build an equivalent representations of all partial solutions recursively over $\mathcal{T}$ using complete patterns only. The operation PREP is then used to create a parity-representation thereof, consisting of CS-patterns only.

▶ **Definition 7.** *Given a pattern p and a label $i \in \text{inc}(p)$, we define the* fix *and* forget *operations, where* $\text{fix}(p, i)$ *adds the singleton* $\{i\}$ *to p, and* $\text{forget}(p, i)$ *removes the label i from all sets of p. Now we define actions* $\text{Ac}(p, \ell)$ *for patterns p with* $|\text{inc}(p)| \leq 2$ *and* $\ell \in [4]$*, by either fixing or forgetting the labels in* $\text{inc}(p)$ *independently.*

It was shown in [8] that $\{\text{fix}(p,i), \text{forget}(p,i)\}$ represents p. Hence, by applying the actions $\text{Ac}(p,\ell)$ for $\ell \in [4]$ on a pattern p with $|\operatorname{inc}(p)| \leq 2$, we get a family of complete patterns that represents p. It is not hard to see that applying pattern operations on CS-patterns yields patterns p with $|\operatorname{inc}(p)| \leq 2$. In particular, $\mathscr{P}_{CS}$ is closed under union and relabeling, whereas for $p \in \mathscr{P}_{CS}$ and $q = \square_{i,j} p$, it holds that $\operatorname{inc}(q) \subseteq \{i,j\}$.

▶ **Definition 8.** *Let $p \in \mathscr{P}_C$. We define $\text{PREP}(p) \subseteq \mathscr{P}_{CS}$ as follows: Let $P_0 = \{p\}$, and let $S_1, \dots S_r$ be all non-singleton sets of p different from Z_p. We define P_i for $i \in [r]$ as follows:*

$$P_i = \bigtriangleup_{q \in P_{i-1}} \Big\{ \big(q \setminus \{Z_q, S_i\}\big) \cup \{Z_q \cup S' \} \colon S' \subset S_i \Big\}.$$

Then we define $\text{PREP}(p) = P_r$. Given a set $S \subseteq \mathscr{P}_C$, we define $\text{PREP}(S) = \bigtriangleup_{p \in S} \text{PREP}(p)$.

Note that each pattern of $\text{PREP}(p)$ results from p by removing all non-singleton sets different from Z_p, and adding some subset of their union to Z_p.

4 The Algorithm

Now we present our main result, namely a dynamic programming algorithm over $\mathcal{T}$. We start by defining the family of states $\mathscr{I}$ that we use to index the dynamic programming tables.

CS-Patterns. We use the family $\mathscr{P}_{CS}$, similar to [8], to count the number of weighted representations of partial solutions in G_x for each $x \in \mathcal{V}$. We note that a CS-pattern p is uniquely defined by its set of labels and its zero-set; Given $X, Y \subseteq U$ the set of labels and the zero-set of p respectively with $Y \subseteq X$, we define $p = \big\{\{u\} \colon u \in X\big\} \cup \big\{\{0\} \cup Y\big\}$.

We define the partial ordering $\preceq_{CS}$ over $\mathscr{P}_{CS}$, where for $p, q \in \mathscr{P}_{CS}$ it holds that $p \preceq_{CS} q$ if and only if $Z_p \subseteq Z_q$ and $\text{label}(p) \subseteq \text{label}(q)$. Clearly, $(\mathscr{P}_{CS}, \preceq_{CS})$ defines a lattice, with $p \vee q = r$ where $\text{label}(r) = \text{label}(p) \cup \text{label}(q)$, and $Z_r = Z_p \cup Z_q$. Hence, the join operation over this lattice corresponds to the union operation over CS-patterns $p \oplus q$.

Colorings and bipartition. We define the set $C_0 = \{\boldsymbol{B}, \boldsymbol{W}\}$, and call it the set of *basic colors*, where $\boldsymbol{B}$ stands for black, and $\boldsymbol{W}$ stands for white. We define the set of *colors* $\overline{C} = \{\emptyset, \boldsymbol{B}, \boldsymbol{W}, \boldsymbol{BW}\}$, where each element of $\overline{C}$ corresponds to a different subset of C_0 in the natural way. We define the ordering $\sqsubseteq_c$ over $\overline{C}$ given by inclusion over the corresponding subsets, namely, given by the two chains $\emptyset \sqsubseteq_c \boldsymbol{B} \sqsubseteq_c \boldsymbol{BW}$ and $\emptyset \sqsubseteq_c \boldsymbol{W} \sqsubseteq_c \boldsymbol{BW}$. We denote by $\sqcup$ the join operation, and by $\sqcap$ the meet operation over the underlying lattice. We call two colors $x, y \in \overline{C}$ *compatible* (denoted by $x \sim y$), if $x \sqcap y = \emptyset$.

We call a mapping $c : U \to \overline{C}$ a *coloring*. We denote the family of all colorings of U by $\mathscr{C} = \overline{C}^U$. We define the ordering $\preceq_C$ over colorings, where for $c_1, c_2 \in \mathscr{C}$, it holds that $c_1 \preceq_C c_2$ if it holds that $c_1(i) \sqsubseteq_c c_2(i)$ for all $i \in U$. Finally, we denote by $\oplus_C$ the join operation over the underlying lattice (pointwise join). For $c \in \mathscr{C}$ and $i, j \in [k]$, we define the coloring $c_{i \to j}$ as $c_{i \to j}(j) = c(i) \sqcup c(j)$, $c_{i \to j}(i) = \emptyset$ and $c_{i \to j}(\ell) = c(\ell)$ for $\ell \notin \{i, j\}$. For $i \in [k]$ and $X \in \overline{C}$, let $c_i^X \in \mathscr{C}$ with $c_i^X(i) = X$ and $c_i^X(\ell) = \emptyset$ for $\ell \neq i$.

Dynamic programming tables. We define the family of *states* $\mathscr{I} = \mathscr{P}_{CS} \times \mathscr{C}$ as the set of combinations of a CS-pattern and a coloring. For $(p, c) \in \mathscr{I}$, the pattern p indicates the connectivity of a partial solution, and the coloring c indicates the *basic colors* assigned to the vertices of each label class in a witness of this partial solution. This coloring allows to extend a partial solution, preserving a proper 2-coloring of the rest of the graph.

We define the ordering $\preceq$ over $\mathscr{I}$ with $(p_1, c_1) \preceq (p_2, c_2)$ if and only if $p_1 \preceq_{CS} p_2$ and $c_1 \preceq_C c_2$. It holds that $\preceq$ is the product of two lattices, and hence, it holds that $(p_1, c_1) \vee (p_2, c_2) = (p, c)$, where $p = p_1 \oplus p_2$ and $c = c_1 \oplus_C c_2$.

Description of the algorithm. Now we describe the algorithm. Intuitively, actions (Definition 7) allow to build (existential) representations of partial solutions by a dynamic programming scheme over $\mathcal{T}$ using complete patterns. However, since the number of complete patterns is at least the number of all partitions of U, we seek to reduce the space of the states by representing (in a count-preserving manner) these patterns using CS-patterns only.

We define the vectors $T\big[x, b, w\big] \in \{0,1\}^{\mathscr{I}}$ for $x \in \mathcal{V}, b \in \overline{B}$ and $w \in \overline{W}$, that constitute the dynamic programming tables of our algorithm, and we show that these vectors can be computed in time $\mathcal{O}^*(12^k)$. In the following sections, we will prove the correctness of the algorithm in two stages. In the first stage we show that any pattern, corresponding to a partial solution at a node x, can be represented by a set of complete patterns resulting from applying different actions at each node in $\mathcal{V}_x^{CJ}$. We assign different weights to the actions at each node resulting in different weights for different representations of partial solutions. In the second stage, we show that the tables $T\big[x, b, w\big]$ count for each CS-pattern p, the number of representations of weight w of solutions of size b that are compatible with p. By appropriately choosing the weight function, we show that this suffices to solve the Connected Odd Cycle Transversal problem with high probability.

▶ **Algorithm 1.** *For $x \in V(\mathcal{T})$, and for two values b, w, we define the vectors $T\big[x, b, w\big] \in \{0,1\}^{\mathscr{I}}$ recursively over $\mathcal{T}$ as follows:*

- *Introduce node $\mu_x = i(v)$: Let $p = [0i]$ if $v = v_0$, and $p = [i]$ otherwise. Let $p_1 = \text{forget}(p, i)$ and $p_2 = \text{fix}(p, i)$. We set all values $T\big[x, b, w\big]\big[q, c\big]$ to zero, for all values of q and c, and then we add one to each of the following entries:*

$$T\big[x, 1, \omega(x,1)\big]\big[(p_1, c_i^{\boldsymbol{\emptyset}})\big] \qquad T\big[x, 1, \omega(x,2)\big]\big[(p_2, c_i^{\boldsymbol{\emptyset}})\big]$$
$$T\big[x, 0, \omega(x,3)\big]\big[([0], c_i^{\boldsymbol{B}})\big] \qquad T\big[x, 0, \omega(x,4)\big]\big[([0], c_i^{\boldsymbol{W}})\big],$$

  where the first two entries correspond to adding the vertex to a partial solution and computing a representation thereof, and the latter two entries correspond to the two different partial colorings of this vertex.
- *Relabel node $\mu_x = \rho_{i\to j}(\mu_{x'})$: For each state $(p, c) \in \mathscr{I}$ we define*

$$T\big[x, b, w\big]\big[(p, c)\big] = \sum_{\substack{(p', c') \in \mathscr{I}, \\ p'_{i\to j} = p \wedge c'_{i\to j} = c}} T\big[x', b, w\big]\big[(p', c')\big].$$

- *Join node $\mu_x = \eta_{i,j}(\mu_{x'})$: For $(p, c) \in \mathscr{I}$, if $c(i) \not\sim c(j)$, we set $T\big[x, b, w\big]\big[(p, c)\big] = 0$. Otherwise, we define*

$$T\big[x, b, w\big]\big[(p, c)\big] = \sum_{\ell \in [4]} \sum_{\substack{p' \in \mathscr{P}_{CS} \\ p \in \text{PREP}\left(\text{Ac}(\square_{i,j} p', \ell)\right)}} T\big[x', b, w - \omega(x, \ell)\big]\big[(p', c)\big].$$

  Intuitively, this corresponds to iterating over all patterns p' at x', applying an add operation to p', and then applying all four actions to the resulting pattern to compute a representation thereof. We add parity-representations of the resulting patterns to $T\big[x, b, w\big]$.
- *Union node $\mu_x = \mu_{x_1} \uplus \mu_{x_2}$: We define*

$$T\big[x, b, w\big]\big[(p, c)\big] = \sum_{\substack{b_1 + b_2 = b \\ w_1 + w_2 = w}} \sum_{\substack{p_1 \oplus p_2 = p \\ c_1 \oplus_C c_2 = c}} T\big[x_1, b_1, w_1\big]\big[(p_1, c_1)\big] \cdot T\big[x_2, b_2, w_2\big]\big[(p_2, c_2)\big].$$

Now we show how to compute these tables in time $\mathcal{O}^*(12^k)$. We assume that an update of a single entry can be done in logarithmic time in the size of the tables (hence, polynomial in n). Although processing a union node x in the naive way requires time polynomial in $(12^k)^2 = 144^k$, we apply a fast convolution technique to compute these tables more efficiently.

In particular, we use a result from [23], which informally states that the $\vee$-product over powers of a finite lattice can be computed efficiently. In the full version, we define a simple lattice over the set $[3] \times [4]$. We show that the lattice $\mathscr{I}$ is isomorphic to the k-th power of this lattice. It follows by the mentioned result that the $\vee$-product over $\mathscr{I}$ can be computed in time $\mathcal{O}^*(12^k)$ implying the same running time (up to a polynomial factor) to process a union node. The running time of the algorithm is a direct consequence of this result, since the tables at a join or a relabel node can be computed in a straight-forward way by iterating over all entries of the tables at a child node and updating the right indices accordingly.

▶ **Corollary 9.** *All tables $T[x, b, w]$ for all values of x, b, w can be computed in time $\mathcal{O}^*(12^k)$.*

5 Solution representation

Now we introduce solution-sequences and action-sequences. A *solution-sequence* is defined by the actions taken at all introduce nodes in a subtree of $\mathcal{T}$ to create a partial solution (together with a witness thereof). We show a bijective mapping between solution-sequences at a node and partial solutions (combined with witnesses) at that node. An *action-sequence* is defined by the actions taken at both introduce and join nodes in a subtree of $\mathcal{T}$, encoding the forget and fix operations along $\mathcal{T}$. Each action-sequence corresponds to a different representation of a partial solution. Intuitively, the set of all action-sequences extending a solution-sequence build a full complete-pattern representation of the corresponding partial solution.

▶ **Definition 10.** *Given a labeled graph G, a set $S \subseteq V(G)$ and a mapping $g\colon S \to \{\boldsymbol{B}, \boldsymbol{W}\}$, we define the coloring $\mathrm{col}^g_G \in \mathscr{C}$ corresponding to g by assigning to each label $\ell \in [k]$ the color defined as follows: let $b = \boldsymbol{B}$ if $\boldsymbol{B} \in g(\mathrm{lab}^{-1}_{G[S]}(\ell))$, and $\emptyset$ otherwise, and let $w = \boldsymbol{W}$ if $\boldsymbol{W} \in g(\mathrm{lab}^{-1}_{G[S]}(\ell))$, and $\emptyset$ otherwise. We define $\mathrm{col}^g_G(\ell) = b \oplus_C w$.*

Intuitively, col^g_G is the coloring defined by assigning to each label ℓ the join of all basic colors assigned to vertices labeled ℓ in S, or $\emptyset$ if no such vertex exists. Hence, it indicates the colors assigned to each label in a witness of a partial solution.

From now on, we shorthand $\mathrm{pat}_x(S)$ for $\mathrm{pat}_{G_x}(S)$, and col^g_x for $\mathrm{col}^g_{G_x}$.

▶ **Definition 11.** *A* solution-sequence *is a mapping $\pi\colon \mathcal{V}^C_x \to \{0, 3, 4\}$. The cost of π is defined as $b(\pi) = |\pi^{-1}(0)|$. The solution-subsequence π_y induced by a node $y \in \mathcal{V}_x$ is the restriction of π to $\mathcal{V}^C_y$. We define the* pattern *$o_\pi = p$ in a natural way, where for an introduce node, if $\pi(x) = 0$ we define $p = [0i]$ if $v = v_0$, and $p = [i]$ if $v \neq v_0$. We define $p = [0]$ otherwise. For different nodes x, p is then defined by applying the corresponding pattern operation to the patterns o_{π_y} for children nodes y of x. We also define the coloring c_π. For an introduce node $i(v)$, we define $c_\pi = C^{\emptyset}_i$ if $i = 0$, $C^{\boldsymbol{B}}_i$ if $\pi(x) = 3$, and $C^{\boldsymbol{W}}_i$ if $\pi(x) = 4$. For a relabel node, we define $c_\pi = (c_{\pi_y})_{i \to j}$. For a join node, we define $c_\pi = c_{\pi_y}$, and for a union node we define $c_\pi = c_{\pi_{y_1}} \oplus_C c_{\pi_{y_2}}$. We say that π is a* valid *solution-sequence, if for each join node $\mu_y = \eta_{i,j,(\mu_{y'})}$, it holds that $c_{\pi_{y'}}(i)$ and $c_{\pi_{y'}}(j)$ are compatible.*

We define an action-sequence *as a mapping $\tau\colon \mathcal{V}^{CJ} \to [4]$ with the cost of τ defined by $b(\tau) = \big|\{y \in \mathcal{V}^C_x \colon \tau(y) \in \{1, 2\}\}\big|$. The weight of τ is defined by $\omega(\tau) = \sum_{y \in \mathcal{V}^{CJ}_x} \omega(y, \tau(y))$. The action-subsequence τ_y at a descendant y of x is defined by the restriction of τ to $\mathcal{V}^{CJ}_y$. Finally, we define the pattern p_τ in a similar way to o_π, where for an introduce node we*

apply a forget operation if $\tau(x) = 1$, and a fix if $\tau(x) = 2$. For a join node, we follow the add operation by the applying the action $\tau(x)$ on the resulting pattern. The coloring c_τ is defined in the same way as c_π where at an introduce node we define replace $\pi(x) = 0$ by $\tau(x) \in \{1, 2\}$. We define a valid *action-sequence in the same way as for solution-sequences. We say that (o_τ, c_τ) is the* pair *generated by τ.*

The notion of valid sequences ensures the validity of witnesses, as it forbids adding edges between vertices of the same color. Now we show the correspondence between solution-sequences and partial solutions at a node x.

▶ **Definition 12.** *Given a set $S \subseteq V_x$ and a mapping $g\colon B_x \setminus S \to C_0$, we define the solution-sequence $\pi_x^{S,g} = \pi$ where for v the vertex introduced at x we have $\pi(x) = 0$ if $v \in S$, $\pi(x) = 3$ if $g(v) = \boldsymbol{B}$, and $\pi(x) = 4$ if $g(v) = \boldsymbol{W}$.*

This forms a bijective mapping between the family of partial solutions at a node x and the family of all solution-sequences at x. In the full version, we show that $b(\pi) = |S|$, $\mathrm{pat}_x(S) = o_\pi$, $\mathrm{col}_x^g = c_\pi$. We also prove that g is a proper 2-coloring of $G_x[V_x \setminus S]$ if and only if π is valid. Now we show that action-sequences represent partial solutions.

▶ **Definition 13.** *A family $R \subseteq \mathscr{P}$* represents *another family S, if it holds for all $q \in \mathscr{P}$ that q is consistent with a pattern of S iff if it is consistent with a pattern of R. The family R* parity-represents *S over a family $C \subseteq \mathscr{P}$, if it holds for each $q \in C$ that q is consistent with an odd number of patterns of S iff it is consistent with an odd number of patterns of R.*

We show in the full version (and from [8]) that pattern operations and actions preserve both representation and parity-representation over $\mathscr{P}_C$, in the sense that if R (parity-) represents S, then $\mathrm{Op}(R)$ (parity-)represents $\mathrm{Op}(S)$. It was also shown in [8] that the four actions over a pattern p with $|\operatorname{inc}(p)| \leq 4$ represent p. Therefore, one can show by induction over $\mathcal{T}$ that the family of patterns corresponding to valid action-sequences at a node x represents the family of patterns corresponding to valid solution-sequences at x, and hence, represents all partial solutions at x as well. The following result follows:

▶ **Corollary 14.** *Let $b \in \overline{B}$. There exist $w \in \overline{W}$, $c \in \mathscr{C}$ and a valid action-sequence τ of cost b and weight w at r generating the pair $([0], c)$, if and only if there exists a connected odd cycle transversal of size b in G containing v_0.*

All that is left to show is that the algorithm correctly counts (modulo 2) the number of valid action-sequences of cost b and weight w at each node x that generate a pair (p, c), where $p \sim q$ for some complete pattern $q \in \mathscr{P}_C$. In order to show this, we prove that $\mathrm{PREP}(p)$ parity-represents p over $\mathscr{P}_C$ for any $p \in \mathscr{P}_C$. Based on this, together with the fact that all defined operations preserve parity-representation over $\mathscr{P}_C$, we prove the following lemma:

▶ **Lemma 15.** *For each node $x \in V(\mathcal{T})$, for all values b, w, and all $c \in \mathscr{C}$ and $q \in \mathscr{P}_C$, it holds that $\sum_{\substack{p \in \mathscr{P}_{CS} \\ p \sim q}} T\big[x, b, w\big]\big[(p, c)\big] \equiv_2 1$, if and only if there exists an odd number of valid action-sequences τ at x of cost b and weight w generating the pair (p, c) with $p \sim q$.*

The following corollary follows directly from Lemma 6:

▶ **Corollary 16.** *It holds for all b, w, and all $c \in \mathscr{C}$ that $T\big[r, b, w\big]\big[([0], c)\big] = 1$ iff there exists an odd number of valid action-sequences of cost b and weight w at r generating $([0], c)$.*

So far, we have shown that the algorithm correctly counts the number of action-sequences of a specific weight, generating a pair $([0], c)$ for any coloring c. We make use of the isolation lemma [35], to show that by a careful choice of the value W and the weight function ω, we get a unique minimum weight sequence with high probability if a solution exists.

▶ **Lemma 17.** *Assume that G contains a connected odd cycle transversal of size $\overline{b}$ containing v_0. Fix $W = 8 \cdot |\mathcal{V}^{CJ}|$, and choose $\omega(x,i)$ independently and uniformly at random in $[W]$. Let τ be a minimum weight valid action-sequence of cost $\overline{b}$ generating a pair $([0],c)$ for any coloring c. Then τ is unique with probability at least $1/2$.*

Proof sketch of Theorem 1. Fix W and ω as in Lemma 17. For each $v \in V$ fix $v_0 = v$, and compute tables $T[x,b,w]$ for all $x \in \mathcal{V}$, and all b, w. The algorithm accepts if there exist a choice of v_0, a coloring $c \in \mathscr{C}$ and a weight w such that $T[r,\overline{b},w]\big[([0],c)\big] = 1$, and rejects otherwise. The algorithms runs in time $\mathcal{O}^*(12^{\mathrm{cw}})$ by Corollary 9.

The correctness follows by the following observation: There exists a solution of cost b if and only if, for any vertex v_0 in this solution, there exists a valid action-sequence at r of cost b generating the pair $([0],c)$ for some $c \in \mathscr{C}$. If such a sequence exists, then by Lemma 17 a unique minimum weight sequence exists with probability at least $1/2$. Hence, it follows from Corollary 16 that if a solution exists, then there exist w and $c \in \mathscr{C}$ such that $T[x,b,w]\big[([0],c)\big] = 1$ with probability at least $1/2$. On the other hand, if no such sequence exists, then it follows by Corollary 16 that $T[x,b,w]\big[([0],c)\big] = 0$ for all values w, any coloring c and any choice of v_0. ◀

6 Lower bound

We prove Theorem 2 through a parameterized reduction from the d-SAT problem for each constant d, proving that an algorithm running in time $\mathcal{O}^*((12-\varepsilon)^{\mathrm{cw}})$ for some $\varepsilon \geq 0$ contradicts SETH. Let I be the given instance with clauses $C_1, \ldots, C_m$ over variables $v_1, \ldots, v_n$. We refer to the full version for details. We start by a formal statement of SETH.

▶ **Conjecture 18** ([27, 28])**.** *For any positive value δ there exists an integer d such that the d-SAT problem cannot be solved in time $\mathcal{O}^*((2-\delta)^n)$, where n is the number of variables.*

Construction of the graph. We build G by combining copies of constant components, called gadgets. We distinguish three kinds of gadgets: *path gadgets*, *decoding gadgets*, and *clause gadgets*. We distinguish a vertex r in G called the *root*, that is forced to every solution, and another vertex $r_{\boldsymbol{B}}$. The connectivity of a solution can be proven by showing that all vertices are connected to r by a path. The vertex $r_{\boldsymbol{B}}$ will not be contained in any optimal solution, and its color serves as reference for a witness of an optimal solution.

Let t_0 be a constant that we fix later, and $s = \lceil n/t_0 \rceil$. We partition the variables into s groups $V_1, \ldots, V_s$ each of size t_0 (except for the last one). Let $t = \lceil t_0 \cdot \log_{12}(2) \rceil$ and $n' = s \cdot t$. The graph G consists of n' path-like structures, called *path sequences*. Each path sequence is a long sequence of path gadgets. Consecutive path gadgets are connected by small bicliques, giving a path sequence its narrow path-like structure. Let $P_1, \ldots, P_{n'}$ be the path sequences of G. We group them into s groups of size t each, calling each of these groups a *path bundle*. For a path bundle $\mathscr{B}_i$ and $j \in [c]$, we call the set of the jth path gadgets on each path sequences in $\mathscr{B}_i$ a *(simple) bundle* B_i^j. Finally, we call the set consisting of the j-th path gadget of each path sequence a *column* $\mathrm{Col}_j = \{X_i^j : i \in [n']\}$. We attach a decoding gadget D_i^j to each bundle and a clause gadget Z_j to each column. See Figure 1 for illustration.

In each path gadget X, We distinguish 3 entry vertices and three exit vertices. We add a biclique between the exit vertices of each path gadget, and the entry vertices of the following path gadget. We attach a special gadget to each entry or exit vertex of a path gadget called a *simple path gadget*. We define the set of *simple states* $\mathscr{I} = \{\boldsymbol{C}, \boldsymbol{D}, \boldsymbol{B}, \boldsymbol{W}\}$. A solution defines a simple state in each simple path gadget. The states defined by the simple path

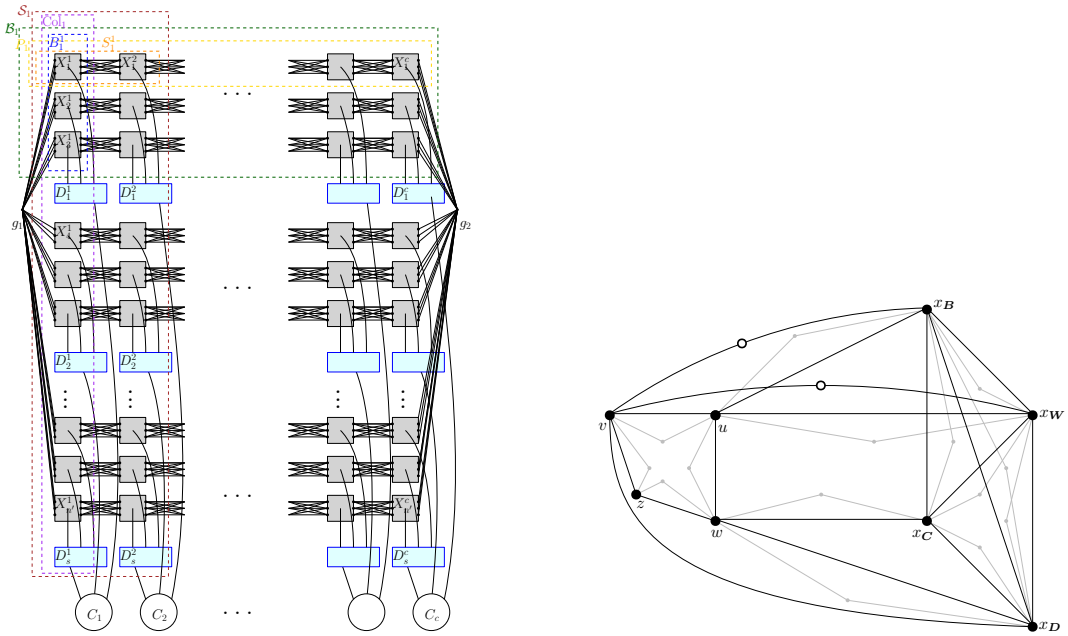


Figure 1 On the left, the graph G corresponding to an instance with two clauses. The gray squares represent path gadgets. Decoding gadgets are depicted in cyan, and clause gadgets as circles. We outline in yellow, blue, green, purple, orange and brown, the first path sequence, bundle, path bundle, column, segment and section respectively. On the right we depict a simple path gadget.

gadgets at the entry vertices of a path gadget combine to one of 12 specific *states* $\mathcal{S} \subseteq \mathscr{I}^3$ (see full version). The combination of states over a bundle B_i^j corresponds to a Boolean assignment over the variable group V_i. The clause gadget and all decoding gadgets on the jth column enforce a partial assignment of some variable group to satisfying a corresponding clause. Finally, the bicliques restrict transitions of states along consecutive path gadgets, ensuring that in any solution there exists a long segment where all path gadgets share the same state along each path. This ensures that the corresponding Boolean assignment satisfies all clauses.

Budget. We choose the value $\overline{b}$ in the output instance $(G, \overline{b})$ by defining a family $\mathcal{F}$ of pairwise vertex-disjoint components in G (mostly the gadgets of G). For each component $X \in \mathcal{F}$, we define a value $b(X)$. We define $\overline{b}$ as the sum of $b(X)$ for all $X \in \mathcal{F}$. In the full version, we show that any solution must contain $b(X)$ vertices in each component $X \in \mathcal{F}$. By the tightness of $\overline{b}$, it follows that a solution of size $\overline{b}$ contains exactly $b(X)$ vertices in each component $X \in \mathcal{F}$, and no other vertices.

Correctness. We show the correctness of our reduction in two steps. First we show that I has a satisfying assignment if and only if G admits a solution of size $\overline{b}$. Second, we show that the graph G has linear clique-width at most $k = n' + k_0$ for some constant k_0. Using these, we show that given an algorithm running in time $\mathcal{O}^*((12-\varepsilon)^k)$ for some $\varepsilon \geq 0$, we can choose the constant t_0 large enough, such that the d-SAT problem for any value of d can be solved in time $\mathcal{O}^*((2-\delta)^n)$ for some positive value δ, contradicting SETH.

7 Conclusion

In this work, we have provided a single-sided error Monte Carlo algorithm for the Connected Odd Cycle Transversal problem with running time $\mathcal{O}^*(12^{cw})$, presenting a new application of the "isolating a representative" technique by [6]. We proved that the running time is tight assuming SETH. This closes the second open problem posed by Hegerfeld and Kratsch [24]. However, one more interesting benchmark problem is still open, namely the Feedback Vertex Set problem. As mentioned in their paper, solving this problem using the Cut&Count technique is challenging, since a direct application would require counting edges induced by partial solutions. Another approach would be to follow the approach of Bojikian and Kratsch [8]. The difficulty of such application lays in finding a low GF(2)-rank representation of partial solutions.

Since both related techniques – the Cut&Count technique [15] and the rank-based approach [5] – were used to solve connectivity problems more efficiently under different structural parameters, it is natural to ask whether the approach of Bojikian and Kratsch [8] can likewise be adapted to different settings. In particular, one would ask whether this results in tight bounds for different connectivity problems under different parameters, where there also exists a gap between the rank of a corresponding compatibility matrix, and the size of a largest triangular submatrix thereof.

References

1 Isolde Adler and Mamadou Moustapha Kanté. Linear rank-width and linear clique-width of trees. *Theor. Comput. Sci.*, 589:87–98, 2015. doi:10.1016/j.tcs.2015.04.021.

2 Benjamin Bergougnoux and Mamadou Moustapha Kanté. Fast exact algorithms for some connectivity problems parameterized by clique-width. *Theor. Comput. Sci.*, 782:30–53, 2019. doi:10.1016/j.tcs.2019.02.030.

3 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets möbius: fast subset convolution. In David S. Johnson and Uriel Feige, editors, *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 67–74. ACM, 2007. doi:10.1145/1250790.1250801.

4 Andreas Björklund, Thore Husfeldt, Petteri Kaski, Mikko Koivisto, Jesper Nederlof, and Pekka Parviainen. Fast zeta transforms for lattices with few irreducibles. *ACM Trans. Algorithms*, 12(1):4:1–4:19, 2016. doi:10.1145/2629429.

5 Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Inf. Comput.*, 243:86–111, 2015. doi:10.1016/j.ic.2014.12.008.

6 Narek Bojikian, Vera Chekan, Falko Hegerfeld, and Stefan Kratsch. Tight bounds for connectivity problems parameterized by cutwidth. In Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté, editors, *40th International Symposium on Theoretical Aspects of Computer Science, STACS 2023, March 7-9, 2023, Hamburg, Germany*, volume 254 of *LIPIcs*, pages 14:1–14:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.STACS.2023.14.

7 Narek Bojikian and Stefan Kratsch. Tight algorithm for connected odd cycle transversal parameterized by clique-width. *CoRR*, abs/2402.08046, 2024. doi:10.48550/arXiv.2402.08046.

8 Narek Bojikian and Stefan Kratsch. A tight monte-carlo algorithm for steiner tree parameterized by clique-width. 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 29:1–29:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ICALP.2024.29.

9 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.

10 Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discret. Appl. Math.*, 101(1-3):77–114, 2000. doi:10.1016/S0166-218X(99)00184-5.

11 Radu Curticapean and Dániel Marx. Tight conditional lower bounds for counting perfect matchings on graphs of bounded treewidth, cliquewidth, and genus. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1650–1669. SIAM, 2016. doi:10.1137/1.9781611974331.ch113.

12 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

13 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.

14 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. *CoRR*, abs/1103.0534, 2011. arXiv:1103.0534.

15 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. *ACM Trans. Algorithms*, 18(2):17:1–17:31, 2022. doi:10.1145/3506707.

16 Baris Can Esmer, Jacob Focke, Dániel Marx, and Pawel Rzazewski. List homomorphisms by deleting edges and vertices: tight complexity bounds for bounded-treewidth graphs. *CoRR*, abs/2210.10677, 2022. doi:10.48550/arXiv.2210.10677.

17 Wolfgang Espelage, Frank Gurski, and Egon Wanke. How to solve np-hard graph problems on clique-width bounded graphs in polynomial time. In Andreas Brandstädt and Van Bang Le, editors, *Graph-Theoretic Concepts in Computer Science, 27th International Workshop, WG 2001, Boltenhagen, Germany, June 14-16, 2001, Proceedings*, volume 2204 of *Lecture Notes in Computer Science*, pages 117–128. Springer, 2001. doi:10.1007/3-540-45477-2_12.

18 Robert Ganian, Thekla Hamm, Viktoriia Korchemna, Karolina Okrasa, and Kirill Simonov. The fine-grained complexity of graph homomorphism parameterized by clique-width. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 66:1–66:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ICALP.2022.66.

19 Carla Groenland, Isja Mannens, Jesper Nederlof, and Krisztina Szilágyi. Tight bounds for counting colorings and connected edge sets parameterized by cutwidth. In Petra Berenbrink and Benjamin Monmege, editors, *39th International Symposium on Theoretical Aspects of Computer Science, STACS 2022, March 15-18, 2022, Marseille, France (Virtual Conference)*, volume 219 of *LIPIcs*, pages 36:1–36:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.STACS.2022.36.

20 Frank Gurski and Egon Wanke. On the relationship between nlc-width and linear nlc-width. *Theor. Comput. Sci.*, 347(1-2):76–89, 2005. doi:10.1016/j.tcs.2005.05.018.

21 Frank Gurski and Egon Wanke. Vertex disjoint paths on clique-width bounded graphs. *Theor. Comput. Sci.*, 359(1-3):188–199, 2006. doi:10.1016/j.tcs.2006.02.026.

22 Falko Hegerfeld and Stefan Kratsch. Towards exact structural thresholds for parameterized complexity. In Holger Dell and Jesper Nederlof, editors, *17th International Symposium on Parameterized and Exact Computation, IPEC 2022, September 7-9, 2022, Potsdam, Germany*, volume 249 of *LIPIcs*, pages 17:1–17:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.IPEC.2022.17.

23 Falko Hegerfeld and Stefan Kratsch. Tight algorithms for connectivity problems parameterized by clique-width. *CoRR*, abs/2302.03627, 2023. doi:10.48550/arXiv.2302.03627.

24 Falko Hegerfeld and Stefan Kratsch. Tight algorithms for connectivity problems parameterized by clique-width. 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 59:1–59:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ESA.2023.59.

25 Falko Hegerfeld and Stefan Kratsch. Tight algorithms for connectivity problems parameterized by modular-treewidth. In Daniël Paulusma and Bernard Ries, editors, *Graph-Theoretic Concepts in Computer Science - 49th International Workshop, WG 2023, Fribourg, Switzerland, June 28-30, 2023, Revised Selected Papers*, volume 14093 of *Lecture Notes in Computer Science*, pages 388–402. Springer, 2023. doi:10.1007/978-3-031-43380-1_28.

26 Pinar Heggernes, Daniel Meister, and Charis Papadopoulos. Characterising the linear clique-width of a class of graphs by forbidden induced subgraphs. *Discret. Appl. Math.*, 160(6):888–901, 2012. doi:10.1016/j.dam.2011.03.018.

27 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.

28 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.

29 Yoichi Iwata and Yuichi Yoshida. On the equivalence among problems of bounded width. In Nikhil Bansal and Irene Finocchi, editors, *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, volume 9294 of *Lecture Notes in Computer Science*, pages 754–765. Springer, 2015. doi:10.1007/978-3-662-48350-3_63.

30 Bart M. P. Jansen and Jesper Nederlof. Computing the chromatic number using graph decompositions via matrix rank. *Theor. Comput. Sci.*, 795:520–539, 2019. doi:10.1016/j.tcs.2019.08.006.

31 Ioannis Katsikarelis, Michael Lampis, and Vangelis Th. Paschos. Structural parameters, tight bounds, and approximation for (k, r)-center. *Discret. Appl. Math.*, 264:90–117, 2019. doi:10.1016/j.dam.2018.11.002.

32 Michael Lampis. Finer tight bounds for coloring on clique-width. *SIAM J. Discret. Math.*, 34(3):1538–1558, 2020. doi:10.1137/19M1280326.

33 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. *CoRR*, abs/1007.5450, 2010. arXiv:1007.5450.

34 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. *ACM Trans. Algorithms*, 14(2):13:1–13:30, 2018. doi:10.1145/3170442.

35 Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Comb.*, 7(1):105–113, 1987. doi:10.1007/BF02579206.

36 Bas A. M. van Geffen, Bart M. P. Jansen, Arnoud A. W. M. de Kroon, and Rolf Morel. Lower bounds for dynamic programming on planar graphs of bounded cutwidth. *J. Graph Algorithms Appl.*, 24(3):461–482, 2020. doi:10.7155/jgaa.00542.

37 Johan M. M. van Rooij, Hans L. Bodlaender, and Peter Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In Amos Fiat and Peter Sanders, editors, *Algorithms - ESA 2009, 17th Annual European Symposium, Copenhagen, Denmark, September 7-9, 2009. Proceedings*, volume 5757 of *Lecture Notes in Computer Science*, pages 566–577. Springer, 2009. doi:10.1007/978-3-642-04128-0_51.

Deterministically Counting k-Paths and Trees Parameterized by Treewidth in Single-Exponential Time

Jonne Visser
Utrecht University, The Netherlands

Hans L. Bodlaender
Utrecht University, The Netherlands

Abstract

In this paper, we give new and faster deterministic algorithms to count the number of k-paths and trees in host graphs of bounded treewidth. Our algorithms use time that is single-exponential in the treewidth, and employ the determinant method from [4]. Modifications of the algorithms count in single-exponential time the number of k-paths between specified end-points, the number of k-cycles, and the number of trees with k vertices that are a subgraph of the host graph.

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

Keywords and phrases Parameterized Complexity, Counting Subgraphs, #k-path, Dynamic Programming, Tree Decomposition, Determinant Method

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.20

1 Introduction

In this paper, we consider the problem of counting subgraphs isomorphic to a graph H in a given graph G. This well-studied problem has several applications, e.g., in bio-informatics when modeling biological networks [1, 12, 13] or when analyzing social networks [8, 17]. The frequency of specific subgraphs may give valuable insights into the functionality of a network and can be used to test the quality of network models.

Counting problems are generally much harder than their *decision* counterparts, asking whether a subgraph H occurs in G at all. Indeed, decision problems with a trivial solution may have a counting variant that is as hard as the counting variant of some NP-complete problems, and many counting problems are $\#P$-Complete [14]. For this reason, most research focuses on the *parameterized* complexity of subgraph counting problems.

To categorize the complexity of intractable parameterized counting problems, a complexity class hierarchy $\#W[t], t \geqslant 1$ analogous to the W-hierarchy of Downey and Fellows was described independently by Flum and Grohe [9] and McCartin [11]. Notably, Flum and Grohe showed that, while deciding whether there is a path of length k in a given graph is FPT, counting k-paths parameterized by k is $\#W[1]$-complete. This makes it unlikely that there is a polynomial time algorithm that counts k-paths even when k is small.

The main contributions of this paper are new and faster algorithms to count respectively paths and trees in graphs of bounded treewidth. First, an $O(18^{tw}k^2n)$ time Dynamic Program for counting the number of k-paths in a host graph G is presented. Although the counting of k-paths parameterized by path length k has been researched a lot [2, 3, 5, 10], exact algorithms parameterized by treewidth have seen little attention. The currently fastest exact algorithm for counting k-paths parameterized by treewidth is a well-known elementary Dynamic Program using a tree decomposition of G to solve the problem in $tw^{O(tw)}poly(k, n)$

© Jonne Visser and Hans L. Bodlaender;
licensed under Creative Commons License CC-BY 4.0

time. The presented Dynamic Program improves on this by providing a *single-exponential time* algorithm for the problem, meaning that a solution is found in $O(c^{O(tw)}poly(k,n))$ time for a constant c.

The second contribution of this paper is an $\ell^{O(tw)}\left(\frac{k}{\ell}\right)^{O(\ell)}n$ time algorithm for $\#k,\ell$-Tree, the problem of counting the number of copies of a tree T with k vertices and ℓ leaves in a host graph G. This result generalizes the k-path counting algorithm in the first part of the paper. Since k-paths are trees, the counting of labeled trees is at least as hard as counting k-paths, showing $\#W[1]$-hardness when parameterized by k. Verstraete and Mubayi proved a lower bound for the number of labeled copies of tree T in host graph G of $(1-\epsilon)nd(d-1)...(d-t+1)$, where n is the number of vertices in G, d is the average degree of vertices in G, t is the number of edges in T and $\epsilon = \frac{(4t)^5}{d^2}$, provided that $d \geqslant t$ [16]. The current fastest exact algorithm to solve $\#k,\ell$-Tree parameterized by treewidth is an elementary Dynamic Program that takes $k^{O(tw)}2^{O(k)}n$ time. The presented algorithm, therefore, improves on this running time when ℓ is significantly smaller than k and even provides a $2^{O(tw)}poly(k,n)$ time algorithm when ℓ is constant.

The techniques presented in this paper build upon the determinant method introduced by Bodlaender et al. in 2015 [4]. This method counts classes of subgraphs that are isomorphic to trees and include a specific vertex, such as Hamiltonian Paths and Steiner Trees. To develop single-exponential time algorithms for $\#k$-Path and $\#k,\ell$-Tree, we relax the constraint of including a specific vertex. This modification enables the counting of subgraphs without restrictions on which vertices are included. Additionally, we demonstrate that the determinant method can be extended to count subgraphs formed by "gluing together" simpler graphs.

Furthermore, the constructed Dynamic Programs can be relatively straightforwardly modified to count the number of fixed end-point k-paths and k-cycles, and to solve $\#k$-Tree (asking for the number of occurrences of *all* trees with k edges in host graph G) in $O(18^{tw}k^4n)$ time. This last problem is proven to be $\#W[1]$-hard when parameterized by k by Brand and Roth [6].

2 Preliminaries

Nice Tree decompositions

In this paper, we utilize nice tree decompositions, following the definition provided by Cygan et al. [7].

Counting trees using determinants

This section explores a determinant-based method for counting subgraphs, developed by Bodlaender et al. [4].

Given an undirected graph $G = (V, E)$ and a nice path or tree decomposition $\mathbb{T}$ of G with width pw or tw respectively, define the *incidence matrix* of G as follows:

▶ **Definition 1** (incidence matrix). *The incidence matrix $A = (a_{v,e})$ of a graph G is an $n \times m$ matrix whose rows are indexed by the vertices in V and whose columns are indexed by the edges in E. Its entries are:*

$$a_{v,e} = \begin{cases} 1, & \text{if } e = (v,w) \text{ and } v > w \\ -1, & \text{if } e = (v,w) \text{ and } v < w \\ 0, & v \notin e, \end{cases} \tag{1}$$

where we use the inherent ordering of vertices in $\mathbb{T}$. Let $V(S)$ be the set of end-points of edges in S; that is, $V(S) = \{v \mid (v, u) \in S\}$. Now, consider the *reduced incidence matrix* of an edge set $S \subseteq E$:

▶ **Definition 2** (reduced incidence matrix). *Let $S \subseteq E$, and $p \in V(S)$. The $(|V(S)| - 1) \times |S|$ matrix $A_p[S]$ is the submatrix of A whose rows are those of A with indices in $V(S) \setminus \{p\}$, and whose columns are those of A with indices in S.*

Leveraging the definition of reduced incidence matrices, we can explore a central lemma in the determinant-based counting method; this lemma is used as a step in the proof for the Matrix Tree Theorem (see, for instance, the combination of Lemmas 2.2 and 2.5 in [15]).

▶ **Lemma 3.** *Let $S \subseteq E$ such that $|S| = |V(S)| - 1$. For any vertex p in $V(S)$, the following holds: if $(V(S), S)$ is a tree, then $|\det(A_p[S])| = 1$. Otherwise, $|\det(A_p[S])| = 0$.*

▶ **Corollary 4.** *Let $S \subseteq E$ such that $|S| = |V(S)| - 1$. If $(V(S), S)$ is a tree, then $\sum_{p \in V(S)} |\det(A_p[S])| = |V(S)|$. Otherwise, $\sum_{p \in V(S)} |\det(A_p[S])| = 0$.*

Corollary 4 allows us to count the number of trees within a collection of edge sets $\mathcal{S} \subseteq \mathcal{P}(E)$ by calculating the value of

$$\sum_{S \in \mathcal{S}} \left(\frac{1}{|V(S)|} \cdot \sum_{p \in V(S)} |\det(A_p[S])| \right), \tag{2}$$

provided that that $|S| = |V(S)| - 1$ for all $S \in \mathcal{S}$.

3 Counting k-paths

Let $G = (V, E)$ be an undirected graph, and let $k \in \mathbf{N}$. We address the following problem:

#k-Path **Input**: A graph $G = (V, E)$ and a parameter $k \in \mathbf{N}$ **Output**: $P_k(G)$; the number of (not necessarily induced) subgraphs of G that are isomorphic to a simple (non-overlapping) path with exactly k vertices.

This section details a Dynamic Programming algorithm for solving the #k-Path problem. Our method first identifies a specific collection of edge sets, $\mathcal{S}$, such that the set of all trees in $\mathcal{S}$ is identical to the set of all simple paths within the graph G. We then utilize $\mathcal{S}$ within Corollary 4 to establish a partial sum constraint that all Dynamic Programming entries must satisfy.

3.1 Partial sum

In line with Corollary 4, we define the collection of edge sets $\mathcal{S}$ as follows:

$$\mathcal{S} = \{S \subseteq E \mid \forall v \in V(S), \deg_S(v) \leqslant 2; |S| = |V(S)| - 1\}.$$

An edge set $S \subseteq E$ constitutes a simple path if and only if the graph $(V(S), S)$ is a tree where every vertex has a degree of at most two. Consequently, the set of trees in $\mathcal{S}$ is identical to the set of simple paths within the graph G. The condition that $|S| = |V(S)| - 1$ for all

$S \in \mathcal{S}$ is inherently satisfied by all trees and thus does not alter the number of trees, but does ensure the compatibility of $\mathcal{S}$ with Corollary 4. The number of simple paths with length k in graph G is now given by

$$\begin{aligned} P_k(G) &= \sum_{S \in \mathcal{S}} [(V(S), S) \text{ is a tree}] \\ &= \frac{1}{k} \sum_{S \in \mathcal{S}, |S|=k-1} \sum_{p \in V(S)} |\det(A_p[S])|. \end{aligned} \tag{3}$$

The condition that $|S| = |V(S)| - 1$, while perhaps intuitive, complicates our Dynamic Programming design by requiring us to track both the number of edges and vertices within any edge set. A more computationally efficient strategy involves counting only the total number of vertices and the number of degree-one vertices. This is particularly advantageous because a simple path invariably has exactly two degree-one vertices. Consequently, we will employ the following proposition:

▶ **Proposition 5.** *Suppose $S \subseteq E$ is a nonempty set such that $deg_S(v) \leq 2$ for all $v \in V(S)$. Then the number of degree one vertices in $V(S)$ is equal to two if and only if $|S| = |V(S)| - 1$.*

With the equivalence from Proposition 5, we can rewrite our definition of $\mathcal{S}$ (without changing $\mathcal{S}$ itself):

$$\mathcal{S} = \{S \subset E \mid \forall v \in V(S), \deg_S(v) \leqslant 2; |\{v \in V(S) \mid \deg_S(v) = 1\}| = 2\}. \tag{4}$$

We can expand the determinant in Equation 3 to obtain

$$P_k(G) = \frac{1}{k} \sum_{S \in \mathcal{S}, |S|=k-1} \sum_{p \in V(S)} \left(\sum_{f : S \overset{1-1}{\to} V(S) \setminus \{p\}} \operatorname{sgn}(f) \prod_{e \in S} a_{f(e),e} \right)^2 \tag{5}$$

$$= \frac{1}{k} \sum_{S \in \mathcal{S}, |S|=k-1} \sum_{p \in V(S)} \sum_{\substack{f_1 : S \overset{1-1}{\to} V(S) \setminus \{p\} \\ f_2 : S \overset{1-1}{\to} V(S) \setminus \{p\}}} \operatorname{sgn}(f_1)\operatorname{sgn}(f_2) \prod_{e \in S} a_{f_1(e),e} a_{f_2(e),e}. \tag{6}$$

The bijection $f : S \overset{1-1}{\to} V(S) \setminus p$ maps each edge $e = (u, v)$ in S to a unique vertex $f(e)$ in $V(S) \setminus p$. Observe that if $f(e)$ is distinct from both u and v, the corresponding term in the sum becomes zero due to the zero entry $a_{f(e),e}$ in the incidence matrix. Utilizing the property that $|\det(A_p[S])| \in 0, 1$, we have $|\det(A_p[S])| = (\det(A_p[S]))^2$.

To compute Equation 6, we will utilize Dynamic Programming on a given tree decomposition $\mathbb{T}$ of G. For this purpose, we define the following parameters:

- For every bag $y \in \mathbb{T}$, $s_{\deg} \in \{0, 1, 2\}^{|B_y|}$ denotes the degree of every vertex in B_y in $G[S]$
- For every bag $y \in \mathbb{T}$, $s_1 \in \{0, 1\}^{|B_y|}$ denotes for every vertex in B_y whether it has been used as a value in the bijection f_1
- For every bag $y \in \mathbb{T}$, $s_2 \in \{0, 1\}^{|B_y|}$ denotes for every vertex in B_y whether it has been used as a value in the bijection f_2
- Let t denote the number of forgotten vertices (in $V_y \setminus B_y$) that have degree one in S
- Let l denote the number of edges in S

Furthermore, we define the following useful sets:

- Let $\mathcal{S}_y(s_{\deg}, t, l)$ be the collection of all edge sets $S \subseteq E_y$ with exactly l edges, with t vertices of degree one in $V(S) \setminus B_y$, and such that the degree of a vertex v in B_y is equal to $s_{\deg}(v)$ and the degree of all vertices is at most two.

- Consider a set of edges $S \subseteq E_y$, and a set of vertices $P \subseteq V(S) \setminus B_y$. For $i \in 1, 2$, we define $\mathcal{F}_y(S, P, s_i)$ as the set of all bijective functions f that map each edge in S to a unique vertex in $V(S) \setminus (s_i^{-1}(0) \cup P)$.
  Intuitively, $\mathcal{F}_y(S, P, s_i)$ represents the possible bijections between the edges in S and a selection of vertices from $V(S)$. Since Proposition 5 states that $|V(S)| > |S|$, not all vertices in $V(S)$ can be part of this bijection. The sets $s_i^{-1}(0)$ and P both identify the vertices that are excluded from this mapping. A key distinction is that $s_i^{-1}(0)$ contains vertices within the bag B_y, whereas P contains vertices that are strictly outside of B_y.
  For a valid k-path we will have that $|P| = 1$ as in Equation 6. However, during the construction of this k-path, we might encounter intermediate states with multiple connected components. In such cases, $|P|$ might need to be greater than one to ensure a valid bijection exists, as more vertices need to be excluded from $V(S)$. Ultimately, we are guaranteed that $|P| = 1$. This is because if $|P| = 0$, the resulting edge sets form cycles, leading to a determinant of zero. Conversely, if $|P| > 1$, the resulting edge sets become disconnected, implying more than two vertices with degree one.

For every entry of the Dynamic Program in bag y we will require that the following partial sum holds:

$$A_y(s_{\deg}, s_1, s_2, t, l) =$$

$$\sum_{S \in \mathcal{S}_y(s_{\deg}, t, l)}^{(1)} \sum_{P \subseteq V(S) \setminus B_y}^{(2)} \sum_{\substack{f_1 \in \mathcal{F}_y(S,P,s_1) \\ f_2 \in \mathcal{F}_y(S,P,s_2)}}^{(3)} \operatorname{sgn}(f_1)\operatorname{sgn}(f_2) \prod_{e \in S}^{(4)} a_{f_1(e),e} a_{f_2(e),e}, \tag{7}$$

where we have labeled the Σ's and Π for later reference. Filling in variables, we see that in a root bag z where $E_z = E$ and $B_z = \emptyset$, we have that $\frac{1}{k} A_z(\emptyset, \emptyset, \emptyset, 2, k-1) = P_k(G)$, the number of k-paths in G.

3.2 Dynamic program

This section details the computation of tables for the functions A_y (defined in Equation 7) for each node type in nice tree decompositions.

Leaf bag y

$$A_y(\emptyset, \emptyset, \emptyset, t, l) = \begin{cases} 1, & \text{if } t = l = 0 \\ 0 & \text{otherwise.} \end{cases} \tag{8}$$

In a leaf bag, we have that $E_y = B_y = \emptyset$. Filling this into Equation 7 we see that Σ (1) has no summands unless $t = l = 0$. In this case, Σ (1) has one summand with $S = \emptyset$, Σ (2) has one summand with $P = \emptyset$, Σ (3) has one summand with $f_1 = f_2 = \emptyset \mapsto \emptyset$ (so that $\operatorname{sgn}(f_1) = \operatorname{sgn}(f_2) = 1$), and Π (4) has no elements.

Introduce vertex u bag y with child z

$$A_y(s_{\deg}, s_1, s_2, t, l) = \begin{cases} A_z(s_{\deg}|_{B_z}, s_1|_{B_z}, s_2|_{B_z}, t, l) & \text{if } s_{\deg}(u) = s_1(u) = s_2(u) \\ & \quad = 0 \\ 0 & \text{otherwise.} \end{cases} \tag{9}$$

Since vertex u is introduced in this bag, no edge in E_y is incident to u. Therefore $\deg_S(u) = 0$ for all $S \in E_y$, and Σ (1) in Equation 7 has no summands if $s_{\deg}(u) \neq 0$. Similarly, all summands in Σ (3) vanish whenever $s_1(u) \neq 0$ or if $s_2(u) \neq 0$.

If $s_{\deg}(u) = s_1(u) = s_2(u) = 0$, then $A_y(s_{\deg}, s_1, s_2, t, l)$ reduces to $A_z(s_{\deg}|_{B_z}, s_1|_{B_z}, s_2|_{B_z}, t, l)$.

Forget vertex u bag y with child z

When $t \in \{1, 2\}$, we set

$$\begin{aligned} A_y(s_{\deg}, s_1, s_2, t, l) = {} & A_z(s_{\deg}[u \leftarrow 0], s_1[u \leftarrow 0], s_2[u \leftarrow 0], t, l) \\ & + A_z(s_{\deg}[u \leftarrow 1], s_1[u \leftarrow 0], s_2[u \leftarrow 0], t-1, l) \\ & + A_z(s_{\deg}[u \leftarrow 1], s_1[u \leftarrow 1], s_2[u \leftarrow 1], t-1, l) \\ & + A_z(s_{\deg}[u \leftarrow 2], s_1[u \leftarrow 0], s_2[u \leftarrow 0], t, l) \\ & + A_z(s_{\deg}[u \leftarrow 2], s_1[u \leftarrow 1], s_2[u \leftarrow 1], t, l). \end{aligned}$$

If $t = 0$, we set

$$\begin{aligned} A_y(s_{\deg}, s_1, s_2, t, l) = {} & A_z(s_{\deg}[u \leftarrow 0], s_1[u \leftarrow 0], s_2[u \leftarrow 0], t, l) \\ & + A_z(s_{\deg}[u \leftarrow 2], s_1[u \leftarrow 0], s_2[u \leftarrow 0], t, l) \\ & + A_z(s_{\deg}[u \leftarrow 2], s_1[u \leftarrow 1], s_2[u \leftarrow 1], t, l). \end{aligned}$$

In a forget vertex u bag y, the vertex u is excluded from B_y. We determine the number of satisfying edge sets by summing over the possible degrees of u.

When $\deg_S(u) = 0$, the edge sets that satisfy Equation 7 are the same as the edge sets in $\mathcal{S}_z(s_{\deg}[u \leftarrow 0], t, l)$.

When $\deg_S(u) = 1$, the edge sets satisfying Equation 7 in bag y are the same as the edge sets in child z that have one less degree one vertex in $S \setminus B_z$. If $u \in P$, A_y reduces to $A_z(s_{\deg}[u \leftarrow 1], s_1[u \leftarrow 0], s_2[u \leftarrow 0], t-1, l)$ if $t \geq 1$, and zero otherwise. When $u \notin P$, A_y reduces to $A_z(s_{\deg}[u \leftarrow 1], s_1[u \leftarrow 1], s_2[u \leftarrow 1], t-1, l)$ when $t \geq 1$, and zero otherwise.

Finally, when $\deg_S(u) = 2$ we see that A_y reduces to $A_z(s_{\deg}[u \leftarrow 2], s_1[u \leftarrow 0], s_2[u \leftarrow 0], t, l)$ when $u \in P$. When $u \notin P$, A_y reduces to $A_z(s_{\deg}[u \leftarrow 2], s_1[u \leftarrow 1], s_2[u \leftarrow 1], t, l)$.

Introduce edge $d = (u, w)$ bag y with child z

For $s_{\deg}(u) > 0$ and $s_{\deg}(w) > 0$ we set

$$\begin{aligned} A_y(s_{\deg}, s_1, s_2, t, l) = {} & A_z(s_{\deg}, s_1, s_2, t, l) + (-1)^{\mathrm{inv}(s_1^{-1}(1), u') + \mathrm{inv}(s_2^{-1}(1), w')} \\ & \times \sum_{\substack{u' \in s_1^{-1}(1) \cap d \\ w' \in s_2^{-1}(1) \cap d}} A_z(s'_{\deg}, s_1[u' \leftarrow 0], s_2[w' \leftarrow 0], t, l-1) \cdot a_{u',d} \cdot a_{w',d}, \end{aligned}$$

where $s'_{\deg}$ is equal to $s_{\deg}$, except that $s'_{\deg}(u) = s_{\deg}(u) - 1$, and $s'_{\deg}(w) = s_{\deg}(w) - 1$. If $s_{\deg}(u) = 0$ or $s_{\deg}(w) = 0$, we can not have that $d \in S$. In this case, we set

$$A_y(s_{\deg}, s_1, s_2, t, l) = A_z(s_{\deg}, s_1, s_2, t, l).$$

In an introduce edge d bag we have two options: either d is included in S, or d is not included in S. If d is not included in S, then A_y reduces to $A_z(s_{\deg}, s_1, s_2, t, l)$.

If $d = (u, w)$ is included in S, we have to account for all possible bijections f_1 and f_2. First note that for all bijections f where $f(d) \notin d$, we have that $a_{f(d),d} = 0$ and the term vanishes. Therefore we fix $f_i(d) = u$ or $f_i(d) = w$.

The inclusion of d into the bijections might change their sign. Since for all $v \in B_y$ and $x \in V_y \setminus B_y$ we have that $v > x$, no vertex in $V_y \setminus B_y$ can contribute to a change in sign. Also, since d is introduced in bag y, we have that $d > e$ for all $e \in E_y \setminus \{d\}$. The change in sign of f_i is therefore equal to $(-1)^{\text{inv}(s_i^{-1}(1), f_i(d))}$.

Join bag y with children z_1 and z_2

$$A_y(s_{\deg}, s_1, s_2, t, l) = \sum_{\substack{s_{\deg,z_1}+s_{\deg,z_2}=s_{\deg} \\ s_{1,z_1}+s_{1,z_2}=s_1 \\ s_{2,z_1}+s_{2,z_2}=s_2 \\ t_{z_1}+t_{z_2}=t \\ l_{z_1}+l_{z_2}=l}} A_{z_1}(s_{\deg,z_1}, s_{1,z_1}, s_{2,z_1}, t_{z_1}, l_{z_1}) A_{z_2}(s_{\deg,z_2}, s_{1,z_2}, s_{2,z_2}, t_{z_2}, l_{z_2}) \cdot (-1)^{\text{inv}(s_{1,z_1}^{-1}(1), s_{1,z_2}^{-1}(1)) + \text{inv}(s_{2,z_1}^{-1}(1), s_{2,z_2}^{-1}(1))}, \tag{10}$$

where we use coordinate-wise addition.

3.3 Running time analysis

Given a host graph G together with a nice path or tree decomposition, we can traverse the decomposition in post-ordering, calculating entries according to the Dynamic Program described above. This procedure gives rise to the following theorem:

▶ **Theorem 6.** *There exists a deterministic algorithm that, given a graph G, a parameter k and a nice tree decomposition of G with treewidth tw, can solve #k-Path in $18^{tw} tw^{O(1)} k^2 n$ time. Given a nice path decomposition with pathwidth pw, there exists an algorithm that can solve #k-Path in $7^{pw} pw^{O(1)} kn$ time.*

4 Counting trees

In this section, we will discuss a generalization of the k-path problem where we count subgraphs isomorphic to a tree T.

Let $G = (V, E)$ be an undirected graph. Then the $\#k, \ell$-Tree problem on G is defined as:

$\#k, \ell$-Tree
Input: A graph $G = (V, E)$ and a tree T that has ℓ leaf nodes and k vertices
Output: $T_k(G)$; the number of (not necessarily induced) subgraphs of G that are isomorphic to T, not accounting for internal symmetries in T.

To count occurrences of a target tree T using Dynamic Programming, we will decompose it into path components. This approach allows us to build upon the path-counting methods detailed in Section 3. However, vertices with a degree of three or more, acting as convergence points for multiple paths, will pose a specific challenge requiring dedicated attention.

4.1 Tree labeling

To count occurrences of a specific tree T, we first need to establish a unique labeling for it. This labeling is derived by decomposing T into path segments, a process that considers three cases based on the degree of each vertex v in T:

- $\deg_T(v) = 1$; v is a **leaf**.
- $\deg_T(v) = 2$; v is a **path node**.
- $\deg_T(v) > 2$; v is an **internal junction**.

We define the path segments as follows:

- Starting in every leaf node with an unlabeled edge, we follow the unique path along the (possibly empty) set of connected path nodes, until we encounter an internal junction j_i or another leaf. The paths traced in this way are called the **leaf paths** of T. Note that if a leaf path terminates in two leaf nodes, T is isomorphic to a simple path, and its occurrences can be counted using the method in Section 3. For the remainder of this paper, we assume all leaf paths end in one leaf and one internal junction.
- For each internal junction, trace the paths starting in its unlabeled edges along the (possibly empty) set of connected path nodes until we encounter another internal junction (note that we can not encounter a leaf node, as per the previous step). These path segments are called **internal paths**.

For an illustration of this procedure, see Figure 1. By design, all edges along paths originating from leaves or internal junctions are labeled. Consequently, since any path not starting at such a point must form a cycle, all edges in the (acyclic) tree T are labeled.

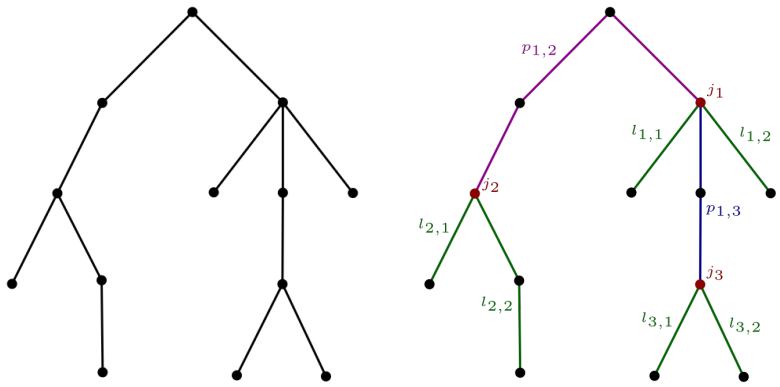


Figure 1 A tree T (left), together with its labeling (right). In the labeled tree, the internal junctions with degree at least 3 are labeled j_1, j_2, j_3, the internal paths are labeled $p_{q,r}$ for a path between j_q and j_r, and the leaf paths are labeled $l_{q,i}$ for the i'th leaf path originating from internal junction j_q.

Let $N = \{j_q\}_q$ be the set of internal junctions in T, let $I = \{p_{q,r} = (j_q, v_1, v_2, ..., j_r), ...\}$ the set of internal paths in T denoted by their vertices, and $L = \{l_{q,i} = (j_q, v_1, v_2, ..., v_k), ...\}$ the set of leaf paths in T denoted by their vertices. We will call (N, I, L) the labeling of T. Using this labeling, we can prove the following theorem, which will help us count the distinct graphs isomorphic to T:

▶ **Theorem 7.** *Let T be a tree with labeling (N, I, L), let $G = (V, E)$ be a graph, and let $S \subseteq E$ be a set of edges in G. Then the subgraph $(V(S), S)$ is isomorphic to T if and only if:*

- *$(V(S), S)$ is a tree*
- *There exists a labeling function $f : V(S) \to N \cup I \cup L$, such that:*

1) ***Internal junctions are the image of exactly one vertex in $V(S)$***
   - *i.e. for all $v \in N$, we have that $|f^{-1}(v)| = 1$*
2) ***Path length is preserved***
   - *i.e. for all internal paths $p_{q,r} \in I$, we have that $|f^{-1}(p_{q,r}) \cup f^{-1}(j_q) \cup f^{-1}(j_r)| = len_T(p_{q,r})$*
   - *and, for all leaf paths $l_{q,i} \in L$, we have that $|f^{-1}(l_{q,i}) \cup f^{-1}(j_q)| = len_T(l_{q,i})$*
3) ***Vertex degrees in paths are preserved***
   - *i.e. for all internal paths $p_{q,r} \in I$, and for all vertices $v \in f^{-1}(p_{q,r})$, we have that $deg_S(v) = 2$.*
   - *and, for all leaf paths $l_q \in L$, there is exactly one vertex $t \in f^{-1}(l_q)$ such that $deg_S(t) = 1$. For all vertices $v \in f^{-1}(l_q) \setminus t$, we have that $deg_S(v) = 2$.*
4) ***Neighborhoods are preserved***
   - *i.e. for all internal paths $p_{q,r} \in I$, and for all vertices $v \in f^{-1}(p_{q,r})$, let $N(v)$ denote the neighborhood of v in S. We have that $f(N(v)) \subseteq \{p_{q,r}, j_q, j_r\}$.*
   - *and, for all leaf paths $l_{q,i} \in L$, and for all vertices $v \in f^{-1}(l_{q,i})$, we have that $f(N(v)) \subseteq \{l_{q,i}, j_q\}$.*

4.2 Partial sum

Consider a graph $G = (V, E)$ and a subset of its edges $S \subseteq E$. Let T be a tree with k vertices, l leaves, and labeling $\mathcal{L}_T = (N, I, L)$. Define $\mathcal{F}_{\mathcal{L}_T}(S)$ as the set of functions $f : V(S) \to N \cup I \cup L$ satisfying the conditions of Theorem 7. If $T(G)$ denotes the number of distinct labeled subgraphs of G isomorphic to T, then Theorem 7 implies that

$$\begin{aligned} T(G) &= \sum_{\substack{S \subseteq E \\ \exists f \in \mathcal{F}_{\mathcal{L}_T}(S)}} [(V(S), S) \text{ is a tree}] \\ &= \frac{1}{k} \sum_{\substack{S \subseteq E \\ |S| = |V(S)| - 1 \\ \exists f \in \mathcal{F}_{\mathcal{L}_T}(S)}} \sum_{p \in V(S)} |\det(A_p[S])|, \end{aligned} \tag{11}$$

where the second step follows from Corollary 4. Consider any edge set S. If there exists a function $f \in \mathcal{F}_{\mathcal{L}_T}(S)$, then by its defining properties (1) and (2), we have:

- $|f^{-1}(N)| = |N|$
- $|f^{-1}(p_{q,r} \cup j_q \cup j_r)| = \text{len}(p_{q,r})$ for all internal paths $p_{q,r}$
- $|f^{-1}(l_{q,i} \cup j_q)| = \text{len}(l_{q,i})$ for all leaf paths $l_{q,i}$

Since every vertex in $V(T)$ is either an internal junction or belongs to an internal/leaf path, it follows that $|V(S)| = |f^{-1}(N \cup I \cup L)| = |V(T)|$. Consequently, the condition $|S| = |V(S) - 1|$ in the first summation simplifies to $|S| = k - 1$. This allows us to expand the determinant in Equation 11 analogously to Section 3, resulting in

$$T(G) = \frac{1}{k} \sum_{\substack{S \subseteq E \\ |S| = k-1 \\ \exists f \in \mathcal{F}_{\mathcal{L}_T}(S)}} \sum_{p \in V(S)} \sum_{\substack{f_1 : S \overset{1-1}{\to} V(S) \setminus \{p\} \\ f_2 : S \overset{1-1}{\to} V(S) \setminus \{p\}}} \text{sgn}(f_1)\text{sgn}(f_2) \prod_{e \in S} a_{f_1(e),e} a_{f_2(e),e}. \tag{12}$$

We now construct the partial sum that defines each entry in our Dynamic Program. The definitions of $s_{\deg}, s_1, s_2$, and $\mathcal{F}_y(S, P, s_i)$ are retained from Section 3. Furthermore, we will define new parameters:

- For every bag $y \in \mathbb{T}$, let $s_a \in (N \cup I \cup L \cup \{0\})^{|B_y|}$ denote the label assigned to every vertex in bag y. Relating to Theorem 7, $s_a(v)$ can be interpreted as $f(v)$ when restricted to the vertices v in B_y.
- Let $\alpha \in \{0,1\}^{|N|}$ denote for every internal junction j_q in N whether some vertex $v \in V(S) \cup B_y$ should be labeled with j_q in the construction of a function $f \in \mathcal{F}_{\mathcal{L}_T}(S)$. The vector α is indexed by the internal junctions in N, and we will denote by $\alpha(j_q)$ the value corresponding to internal junction j_q. Conversely, we will denote by $\alpha^{-1}(1)$ all internal junctions in N for which α is equal to one.
- Let $\lambda \in \mathbb{N}^{|I \cup L|}$ denote for every internal path in I and leaf path in L how many vertices in $V(S)$ should be labeled with this path in the construction of a function $f \in \mathcal{F}_{\mathcal{L}_T}(S)$. The vector λ is indexed by the internal and leaf paths in $I \cup L$, and we will denote by $\lambda(p)$ the value corresponding to the path p. Conversely, we will denote by $\lambda^{-1}(\mathbb{N}_{\geqslant 1})$ all paths in $I \cup L$ for which λ is at least one.
- Let $t_{leaf} \in \{0,1\}^{|L|}$ denote for every leaf path $l_{q,i}$ in L the number of forgotten vertices v with degree one for which $f(v)$ should map to $l_{q,i}$ in the construction of some $f \in \mathcal{F}_{\mathcal{L}_T}(S)$.
- Let $\mathcal{S}_y(s_{\deg}, s_a, \alpha, \lambda, t_{leaf})$ be the collection of all edge sets $S \subseteq E_y$ such that there exists a labeling function $f_{y,S} : V(S) \cup B_y \to \alpha^{-1}(1) \cup \lambda^{-1}(\mathbb{N}_{\geqslant 1}) \cup \{0\}$ that obeys the following requirements:

  0) ***The functions $f_{y,S}$ and s_a are identical when restricted to bag vertices***
  - i.e. for all bag vertices $v \in B_y$, we have that $f_{y,S}(v) = s_a(v)$

  1) ***All internal junctions in $\alpha^{-1}(1)$ are the image of exactly one vertex in $V(S) \cup B_y$ under $f_{y,S}$***
  - i.e. for all internal junctions $v \in \alpha^{-1}(1)$, we have that $|f_{y,S}^{-1}(v)| = 1$

  2) ***All paths p in $I \cup L$ are the image of exactly $\lambda^{-1}(p)$ vertices in $V(S) \cup B_y$ under $f_{y,S}$***
  - i.e. for all paths $p \in \lambda^{-1}(\mathbb{N}_{\geqslant 1})$, we have that $|f_{y,S}^{-1}(p)| = \lambda(p)$

  3) ***Vertex degrees are preserved in forgotten vertices and are equal to s_{deg} in bag vertices***
  - i.e. for all internal junctions $j_q \in \alpha^{-1}(1)$, if the vertex $v \in f_{y,S}^{-1}(j_q)$ (there is exactly one such vertex as per (1)) is a forgotten vertex, then $\deg_S(v) = \deg_T(j_q)$. If v is a bag vertex, then $\deg_S(v) = s_{\deg}(v)$.
  - for all internal paths $p_{q,r} \in \lambda^{-1}(\mathbb{N}_{\geqslant 1}) \cap I$, we have that all forgotten vertices $v \in f_{y,S}^{-1}(p_{q,r}) \setminus B_y$ have degree two in S. All bag vertices $v \in f_{y,S}^{-1}(p_{q,r}) \cap B_y$ have degree $s_{\deg}(v)$ in S.
  - and, for all leaf paths $l_{q,i} \in \lambda^{-1}(\mathbb{N}_{\geqslant 1}) \cap L$, we have that exactly $t_{leaf}(l_{q,i})$ forgotten vertices in $f_{y,S}^{-1}(l_{q,i}) \setminus B_y$ have degree one in S, and all other vertices in $f_{y,S}^{-1}(l_{q,i}) \setminus B_y$ have degree two in S. All bag vertices $v \in f_{y,S}^{-1}(l_{q,i}) \cap B_y$ have degree $s_{\deg}(v)$ in S.

  4) ***Neighborhoods are preserved***
  - i.e. for all internal paths $p_{q,r} \in \lambda^{-1}(\mathbb{N}_{\geqslant 1}) \cap I$, and for all vertices $v \in f_{y,S}^{-1}(p_{q,r})$, let $N(v)$ denote the neighborhood of v in S. We have that $f_{y,S}(N(v)) \subseteq \{p_{q,r}, j_q, j_r\}$.
  - and, for all leaf paths $l_{q,i} \in \lambda^{-1}(\mathbb{N}_{\geqslant 1}) \cap L$, and for all vertices $v \in f_{y,S}^{-1}(l_{q,i})$, we have that $f_{y,S}(N(v)) \subseteq \{l_{q,i}, j_q\}$.

With the parameters defined above, we will determine the Dynamic Program entries A_y for each bag y such that

$$A_y(s_{\deg}, s_1, s_2, s_a, \alpha, \lambda, t_{leaf}) = \sum_{S \in \mathcal{S}_y(s_{\deg}, s_a, \alpha, \lambda, t_{leaf})}^{(1)} \sum_{P \subseteq V(S) \setminus B_y}^{(2)} \sum_{\substack{f_1 \in \mathcal{F}_y(S,P,s_1) \\ f_2 \in \mathcal{F}_y(S,P,s_2)}}^{(3)} \operatorname{sgn}(f_1)\operatorname{sgn}(f_2) \prod_{e \in S}^{(4)} a_{f_1(e),e} a_{f_2(e),e}. \tag{13}$$

Consider a vector $\lambda_T \in \mathbb{N}^{|I \cup L|}$ indexed by all internal paths (I) and leaf paths (L) of $\mathcal{L}_T$. For each internal path $p \in I$, the corresponding value in λ_T is $\operatorname{len}(p) - 2$, and for each leaf path $p \in L$, the value is $\operatorname{len}(p) - 1$. Then, in a bag y with $E_y = E$ and $B_y = \emptyset$, the expression $\frac{1}{k} A_y(\emptyset, \emptyset, \emptyset, \emptyset, 1^{|N|}, \lambda_T, 1^{|L|})$ precisely equals Equation 12, thus yielding the number of subgraphs in G isomorphic to T.

4.3 Dynamic program

In this section, we will describe a Dynamic Program that iteratively calculates all partial sums of Equation 13. The entry calculations are accompanied by a brief explanation.

Leaf Bag y

$$A_y(\emptyset, \emptyset, \emptyset, \emptyset, \alpha, \lambda, t_{leaf}) = \begin{cases} 1, & \text{if } \alpha = 0^{|N|}, \lambda = 0^{|I \cup L|}, t_{leaf} = 0^{|L|}, \\ 0 & \text{otherwise.} \end{cases} \tag{14}$$

Introduce vertex u bag y with child z

$$A_y(s_{\deg}, s_1, s_2, s_a, \alpha, \lambda, t_{leaf}) = \begin{cases} A_z(s_{\deg}|_{B_z}, s_1|_{B_z}, s_2|_{B_z}, s_a|_{B_z}, \alpha, \lambda, t_{leaf}) & \text{if } s_{\deg}(u) = s_1(u) = s_2(u) = 0, \\ & s_a(u) = \emptyset, \\ A_z(s_{\deg}|_{B_z}, s_1|_{B_z}, s_2|_{B_z}, s_a|_{B_z}, & \text{if } s_{\deg}(u) = s_1(u) = s_2(u) = 0, \\ \qquad \alpha[j_q \leftarrow 0], \lambda, t_{leaf}) & s_a(u) = j_q \in N, \alpha(j_q) = 1, \\ A_z(s_{\deg}|_{B_z}, s_1|_{B_z}, s_2|_{B_z}, s_a|_{B_z}, \alpha, & \text{if } s_{\deg}(u) = s_1(u) = s_2(u) = 0, \\ \qquad \lambda[p \leftarrow \lambda(p) - 1], t_{leaf}) & s_a(u) = p \in I \cup L, \lambda(p) > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{15}$$

Because vertex u is introduced in this bag, no edge in E_y is incident to u, and therefore the partial sum reduces to zero if $s_{\deg}(u) \neq 0, s_1(u) \neq 0$ or $s_2(u) \neq 0$. Depending on the label $s_a(u)$ there are slightly different cases.

Forget vertex u bag y with child z

$$
\begin{aligned}
&A_y(s_{\deg}, s_1, s_2, s_a, \alpha, \lambda, t_{leaf}) = \\
&\quad A_z(s_{\deg}[u \leftarrow 0], s_1[u \leftarrow 0], s_2[u \leftarrow 0], s_a[u \leftarrow 0], \alpha, \lambda, t_{leaf}) \\
&+ \sum_{p \in t_{leaf}^{-1}(1)} \begin{pmatrix} A_z(s_{\deg}[u\leftarrow 1], s_1[u\leftarrow 0], s_2[u\leftarrow 0], s_a[u\leftarrow p], \alpha, \lambda, t_{leaf}[p\leftarrow 0]) \\ +A_z(s_{\deg}[u\leftarrow 1], s_1[u\leftarrow 1], s_2[u\leftarrow 1], s_a[u\leftarrow p], \alpha, \lambda, t_{leaf}[p\leftarrow 0]) \end{pmatrix} \\
&+ \sum_{p \in \lambda^{-1}(\mathbb{N}_{\geqslant 1})} \begin{pmatrix} A_z(s_{\deg}[u\leftarrow 2], s_1[u\leftarrow 0], s_2[u\leftarrow 0], s_a[u\leftarrow p], \alpha, \lambda, t_{leaf}) \\ +A_z(s_{\deg}[u\leftarrow 2], s_1[u\leftarrow 1], s_2[u\leftarrow 1], s_a[u\leftarrow p], \alpha, \lambda, t_{leaf}) \end{pmatrix} \\
&+ \sum_{j_q \in \alpha^{-1}(1)} \begin{pmatrix} A_z(s_{\deg}[u\leftarrow \deg_T(j_q)], s_1[u\leftarrow 0], s_2[u\leftarrow 0], s_a[u\leftarrow j_q], \alpha, \lambda, t_{leaf}) \\ +A_z(s_{\deg}[u\leftarrow \deg_T(j_q)], s_1[u\leftarrow 1], s_2[u\leftarrow 1], s_a[u\leftarrow j_q], \alpha, \lambda, t_{leaf}) \end{pmatrix}. \qquad (16)
\end{aligned}
$$

To obtain the total number of partial solutions in a forget bag we sum over all possible degrees of u in B_z (denoted as $s_{\deg,z}(u)$). The case that $s_{\deg,z}(u) = 0$ accounts for the edge sets S such that $u \notin S$. The case that $s_{\deg,z}(u) = 1$ accounts for the edge sets where u is a leaf, the case that $s_{\deg,z}(u) = 2$, accounts for the edge sets where u is a path node, and the case where $s_{\deg,z}(u) > 2$, accounts for the edge sets where u is an internal junction.

Introduce edge $d = (u, w)$ bag y with child z

Due to the preservation of neighborhoods requirement (4) for $f_{y,S}$, the edge d can only be included in S if there is some internal path $p_{q,r} \in I$ such that $s_a(u), s_a(w) \in \{p_{q,r}, j_q, j_r\}$, or if there is some leaf path $l_{q,i}$ such that $s_a(u), s_a(w) \in \{l_{q,i}, j_q\}$. Let $1_{I\cup L}(u, w)$ be the indicator function that is equal to one if there exists such a path, and zero otherwise. If $s_{\deg}(u) = 0$, $s_{\deg}(w) = 0$, or if $1_{I\cup L}(u, w) = 0$, we set

$$
A_y(s_{\deg}, s_1, s_2, s_a, \alpha, \lambda, t_{leaf}) = A_z(s_{\deg}, s_1, s_2, s_a, \alpha, \lambda, t_{leaf}), \qquad (17)
$$

Otherwise, we set

$$
\begin{aligned}
A_y(s_{\deg}, s_1, s_2, s_a, \alpha, \lambda, t_{leaf}) &= A_z(s_{\deg}, s_1, s_2, s_a, \alpha, \lambda, t_{leaf}) \\
&+ \sum_{\substack{u' \in s_1^{-1}(1) \cap d \\ w' \in s_2^{-1}(1) \cap d}} \begin{pmatrix} A_z(s'_{\deg}, s_1[u'\leftarrow 0], s_2[w'\leftarrow 0], s_a, \alpha, \lambda, t_{leaf}) \\ \cdot a_{u',d} \cdot a_{w',d} \cdot (-1)^{\mathrm{inv}(s_1^{-1}(1), u') + \mathrm{inv}(s_2^{-1}(1), w')} \end{pmatrix}. \qquad (18)
\end{aligned}
$$

In an introduce edge d bag we have two options: either d is included in S, or it is not. In the case that $d \notin S$, then $A_y(s_{\deg}, s_1, s_2, s_a, \alpha, \lambda, t_{leaf})$ reduces to $A_z(s_{\deg}, s_1, s_2, s_a, \alpha, \lambda, t_{leaf})$. If $d \in S$ we have to correctly account for the change in sign of bijections f_1 and f_2. This is done in the same way as in Section 3.2.

Join bag y with children z_1 and z_2

$$
\begin{aligned}
&A_y(s_{\deg}, s_1, s_2, s_a, \alpha, \lambda, t_{leaf}) = \\
&\sum_{\substack{s_{\deg,z_1} + s_{\deg,z_2} = s_{\deg} \\ s_{a,z_1} = s_{a,z_2} = s_a \\ \forall j_q \in N, \alpha_{z_1}(j_q) + \alpha_{z_2}(j_q) = \alpha(j_q) + |s_a^{-1}(j_q)| \\ \forall p \in I \cup L, \lambda_{z_1}(p) + \lambda_{z_2}(p) = \lambda(p) + |s_a^{-1}(p)| \\ t_{leaf,z_1} + t_{leaf,z_2} = t_{leaf} \\ s_{1,z_1} + s_{1,z_2} = s_1 \\ s_{2,z_1} + s_{2,z_2} = s_2}} \begin{matrix} A_{z_1}(s_{\deg,z_1}, s_{1,z_1}, s_{2,z_1}, s_{a,z_1}, \alpha_{z_1}, \lambda_{z_1}, t_{leaf,z_1}) \\ \cdot A_{z_2}(s_{\deg,z_2}, s_{1,z_2}, s_{2,z_2}, s_{a,z_2}, \alpha_{z_2}, \lambda_{z_2}, t_{leaf,z_2}) \\ \cdot (-1)^{\mathrm{inv}(s_{1,z_1}^{-1}(1), s_{1,z_2}^{-1}(1)) + \mathrm{inv}(s_{2,z_1}^{-1}(1), s_{2,z_2}^{-1}(1))}, \end{matrix} \qquad (19)
\end{aligned}
$$

where we use coordinate-wise addition.

4.4 Running time analysis

Using the Dynamic Program above we can prove the following theorem:

▶ **Theorem 8.** *There exists a deterministic algorithm that, given a graph G together with a nice tree decomposition of G with treewidth tw and given a tree T with ℓ leaf nodes and k vertices with $\ell < \frac{k-1}{2e}$, can solve $\#k, \ell$-Tree in $\ell^{O(tw)} tw^{O(1)} \left(\frac{k}{\ell}\right)^{O(\ell)} n$ time. Given a nice path decomposition of G with pathwidth pw, there exists an algorithm that solves $\#k, \ell$-Tree $\ell^{O(pw)} pw^{O(1)} \left(\frac{k}{\ell}\right)^{O(\ell)} n$ time.*

5 Conclusion

In this paper, we have given deterministic algorithms for the exact counting of k-paths and trees. These algorithms are single-exponential, parameterized by the tree- or pathwidth of the host graph G, and are linear in the number of vertices in G. Both algorithms utilize the determinant-based approach introduced by Bodlaender et al. [4] to construct a Dynamic Program. The algorithms in this paper can also be straightforwardly modified to count the number of cycles of length k, the number of fixed-endpoint k-paths, and the number of k-Trees in single-exponential time when parameterized by the treewidth of the host graph.

Before our work, no algorithms were known for the studied problems with single-exponential dependency on the treewidth. However, the base of the exponents of the algorithm to count trees is rather large. An interesting question is whether these bases can be improved. One possible way in which the algorithm could be improved is if a more efficient way of counting the length of the internal and leaf paths is found. Keeping track of how many edges are in each path quickly blows up the running time, as there are $\left(\frac{k}{\ell}\right)^{O(\ell)}$ possible combinations. A more efficient way of counting would significantly reduce the scaling of the running time with ℓ.

The techniques of this paper can be used to count even more general graph patterns. Suppose for example that some graph H is 'almost' acyclic. That is, there is some small set of edges C that, when removed, H becomes acyclic. Then we can treat the set of end-points $V(C)$ of those edges in C as internal junctions (so we can label vertices as end-points of such edges), and we can verify the existence of edges in G between those end-points without including them in the bijections. The running time of such an algorithm would be similar to that of the tree counting algorithm but with ℓ replaced with $\ell + |V(C)|$.

References

1 Noga Alon, Phuong Dao, Iman Hajirasouliha, Fereydoun Hormozdiari, and S. Cenk Sahinalp. Biomolecular network motif counting and discovery by color coding. *Bioinformatics*, 24(13): i241–i249, 2008. doi:10.1093/bioinformatics/btn163.

2 Noga Alon and Shai Gutner. Balanced hashing, color coding and approximate counting. In *Proceedings 4th International Workshop on Parameterized and Exact Computation, IWPEC 2009*, pages 1–16. Springer, 2009. doi:10.1007/978-3-642-11269-0_1.

3 V. Arvind and Venkatesh Raman. Approximation algorithms for some parameterized counting problems. In *Proceedings 13th International Symposium on Algorithms and Computation, ISAAC 2002*, pages 453–464. Springer, 2002. doi:10.1007/3-540-36136-7_40.

4 Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Information and Computation*, 243:86–111, 2015. doi:10.1016/j.ic.2014.12.008.

5 Cornelius Brand, Holger Dell, and Thore Husfeldt. Extensor-coding. In *ACM SIGACT 50th Annual Symposium on Theory of Computing, STOC 2018*, pages 151–164, 2018. doi:10.1145/3188745.3188902.

6 Cornelius Brand and Marc Roth. Parameterized counting of trees, forests and matroid bases. In *12th International Computer Science Symposium in Russia: Computer Science – Theory and Applications, CSR 2017*, pages 85–98. Springer, 2017. doi:10.1007/978-3-319-58747-9_10.

7 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M.M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. *ACM Transactions on Algorithms*, 18(2):17:1–17:31, 2022. doi:10.1145/3506707.

8 Alexandra Duma and Alexandru Topirceanu. A network motif based approach for classifying online social networks. In *Proceedings 9th IEEE International Symposium on Applied Computational Intelligence and Informatics, SACI 2014,*, pages 311–315. IEEE, 2014. doi:10.1109/SACI.2014.6840083.

9 Jörg Flum and Martin Grohe. The parameterized complexity of counting problems. *SIAM Journal on Computing*, 33(4):892–922, 2004. doi:10.1137/S0097539703427203.

10 Daniel Lokshtanov, Andreas Björklund, Saket Saurabh, and Meirav Zehavi. Approximate counting of k-paths: Simpler, deterministic, and in polynomial space. *ACM Trans. Algorithms*, 17(3), 2021. doi:10.1145/3461477.

11 Catherine McCartin. Parameterized counting problems. *Annals of Pure and Applied Logic*, 138(1):147–182, 2006. doi:10.1016/j.apal.2005.06.010.

12 R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network Motifs: Simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002. doi:10.1126/science.298.5594.824.

13 N. Pržulj, D. G. Corneil, and I. Jurisica. Modeling interactome: scale-free or geometric? *Bioinformatics*, 20(18):3508–3515, July 2004. doi:10.1093/bioinformatics/bth436.

14 L.G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979. doi:10.1016/0304-3975(79)90044-6.

15 Janneke van den Boomen. The Matrix Tree Theorem, 2007. Bachelor thesis, Radbout Universiteit Nijmegen. URL: https://www.math.ru.nl/~bosma/Students/jannekebc3.pdf.

16 Jacques Verstraete and Dhruv Mubayi. Counting Trees in Graphs. *The Electronic Journal of Combinatorics*, 23(3):P3.39, 2016. doi:10.37236/6084.

17 Stanley Wasserman and Katherine Faust. *Social Network Analysis: Methods and Applications*, volume 8. Cambridge University Press, 1994. doi:10.1017/CBO9780511815478.

A Polynomial Delay Algorithm Generating All Potential Maximal Cliques in Triconnected Planar Graphs

Alexander Grigoriev
Department of Data Analytics and Digitalisation, Maastricht University, The Netherlands

Yasuaki Kobayashi
Hokkaido University, Sapporo, Japan

Hisao Tamaki
Meiji University, Kawasaki, Japan

Tom C. van der Zanden
Department of Data Analytics and Digitalisation, Maastricht University, The Netherlands

Abstract

We develop a new characterization of potential maximal cliques of a triconnected planar graph and, using this characterization, give a polynomial delay algorithm generating all potential maximal cliques of a given triconnected planar graph. Combined with the dynamic programming algorithm due to Bouchitté and Todinca, this algorithm leads to a treewidth algorithm for general planar graphs that runs in time linear in the number of potential maximal cliques and polynomial in the number of vertices.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Graph algorithms analysis

Keywords and phrases potential maximal cliques, treewidth, planar graphs, triconnected planar graphs, polynomial delay generation

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.21

Funding *Yasuaki Kobayashi*: JSPS KAKENHI Grant Numbers JP23K28034, JP24H00686 and JP24H00697
Hisao Tamaki: JSPS KAKENHI Grant Number JP24H00697

1 Introduction

Let G be a graph. A vertex set X of G is a *potential maximal clique* (PMC) of G if there is a minimal triangulation H of G such that X is a maximal clique of H. Let $\Pi(G)$ denote the set of all PMCs of G. Computing $\Pi(G)$ is the first step in the treewidth algorithms of Bouchitté and Todinca [7]. The second step is a dynamic programming algorithm that works on $\Pi(G)$ and computes the treewidth $tw(G)$ of G, together with a tree-decomposition of G achieving the width, in time $|\Pi(G)|n^{O(1)}$. The same approach works for various problems that can be formulated as asking for an optimal triangulation with some criterion, including the minimum fill-in problem [7]. See [11] and [12] for more applications of PMCs in this direction.

Naturally, the complexity of computing $\Pi(G)$ is of great interest. In a separate paper [8], Bouchitté and Todinca showed that $\Pi(G)$ can be computed in time polynomial in the number of minimal separators (see Section 2 for the definition) of G. Fomin and Villanger [13] showed that $|\Pi(G)|$ is $O(1.7549^n)$ and gave an algorithm to compute $\Pi(G)$ in time $O(1.7549^n)$. (These bounds were later improved to $O(1.7346^n)$ in [12].) Although this bound on the running time matches the combinatorial bound, an algorithm with running time $|\Pi(G)|n^{O(1)}$

© Alexander Grigoriev, Yasuaki Kobayashi, Hisao Tamaki, and Tom C. van der Zanden;
licensed under Creative Commons License CC-BY 4.0

is much more desirable in practice. In fact, the success of recent practical algorithms for treewidth based on the Bouchitté-Todinca algorithm, [20] for example, relies on the fact that $|\Pi(G)|$ is vastly smaller than this theoretical bound on many instances of interest in practice.

In this paper, we address this question if $\Pi(G)$ can be computed in time $|\Pi(G)|n^{O(1)}$, which is one of the open questions in parameterized and exact computation listed in [4]. We note that this question has been open for any natural graph class except those for which $\Pi(G)$ is known to be computable in polynomial time. We answer this question in the affirmative for the class of triconnected planar graphs. As a straightforward consequence, we obtain a new running time upper bound of $|\Pi(G)|n^{O(1)}$ on the treewidth computation, not only for triconnected planar graphs but also for general planar graphs, since the treewidth of a planar graph can be computed by working separately on its triconnected components.

This result is also interesting with regard to a broader question: how planarity helps in treewidth computation? In the case of branchwidth, a graph parameter similar to treewidth, there is a clear answer to the similar question. Branchwidth is NP-hard to compute for general graphs but polynomial time computable for planar graphs by the celebrated Ratcatcher algorithm due to Seymour and Thomas [19]. It has been a long open question whether an analogy holds for treewidth: we do not know whether computing treewidth is NP-hard or polynomial time solvable on planar graphs. In addition to the 1.5-approximation algorithm based on the Ratcatcher algorithm for branchwidth, several approximation algorithms for treewidth are known that exploit planarity ([17] for example, also see citations therein), but no non-trivial exact algorithm for planar treewidth is known. Several fixed-parameter tractable exact treewidth algorithms for general graphs are known, including a $2^{O(k^3)}n$ time algorithm [3] and a $2^{O(k^2)}n^{O(1)}$ time algorithm [18]. No improvement, however, on these results exploiting planarity is known.

Our algorithm is the first one that explicitly exploits planarity for computing exact treewidth in a non-trivial manner.

Our algorithm not only computes $\Pi(G)$ for triconnected planar graph G in time $|\Pi(G)|n^{O(1)}$ but also runs with polynomial delay: it spends $n^{O(1)}$ time for the generation of each element in the set. See Section 4 for a more technical definition of polynomial delay generation. This feature of our algorithm is appealing in the field of combinatorial enumeration where designing an algorithm with polynomial delay is one of the main research goals. We note that most of the devices we use in our algorithm are still needed even if we downgrade our requirement of polynomial delay to the total running time bound of $|\Pi(G)|n^{O(1)}$.

Our generation algorithm is based on a new and simple characterization of PMCs of triconnected planar graphs. Fomin, Todinca, and Villanger [10] used a characterization of PMCs of general planar graphs in terms of what they call *θ-structures*. A θ-structure is, roughly speaking, a collection of three curves sharing their ends in the sphere of embedding such that each pair forms a *noose* of the embedded graph, where a noose is a simple closed curve that intersects the embedded graph only at vertices. They used this characterization to design an algorithm for the maximum induced planar subgraph problem. Unfortunately for our purposes, this characterization does not seem to lead to an efficient algorithm for computing $\Pi(G)$ of planar graphs.

The simplicity of our characterization comes from the use of what we call a *latching graph* of a triconnected plane graph instead of the traditional tools for reasoning about separators in plane graphs: nooses or radial graphs, which are bipartite plane graphs representing the incidences of the vertices and faces (for a precise definition, see [16] for example). The latching graph L_G of a biconnected plane graph G is a multigraph obtained from G by

adding, in each face of G, every chord of the bounding cycle of the face. We observe that L_G is simple if G is triconnected (Proposition 6). Separators of G correspond to cycles of L_G in a similar way as they correspond to cycles of the radial graph of G. Unlike a radial graph, however, that is a bipartite graph on the vertices and the faces of the embedded graph G, a latching graph is a graph on $V(G)$ and simpler to reason with for some purposes. More specifically, we define a class of biconnected plane graphs we call *steerings* (Definition 13 in Section 3) and show that a vertex set X of a triconnected plane graph G is a PMC if and only if the subgraph of L_G induced by X is a steering. This characterization would be less simple if it was formulated in terms of radial graphs or nooses. It is not clear if the latching multigraph is useful for reasoning with plane graphs that are not triconnected. Adding this notion of latching graphs to the toolbox for triconnected planar graphs is one of our contributions.

The proofs of propositions and lemmas marked with $\star$ will be found in the journal version of this paper.

2 Preliminaries

Graphs. In this paper, all graphs of main interest are simple, that is, without self loops or parallel edges. Let G be a graph. We denote by $V(G)$ the vertex set of G and by $E(G)$ the edge set of G. When the vertex set of G is V, we say that the graph G is *on* V. The *complete graph* on V, denoted by $K(V)$, is a graph on V where every vertex is adjacent to all other vertices. A complete graph $K(V)$ is a K_n if $|V| = n$.

A graph H is a *subgraph* of a graph G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. The subgraph of G induced by $U \subseteq V(G)$ is denoted by $G[U]$: its vertex set is U and its edge set is $\{\{u, v\} \in E(G) \mid u, v \in U\}$. A vertex set $C \subseteq V(G)$ is a *clique* of G if $G[C]$ is a complete graph. For each $v \in V(G)$, $N_G(v)$ denotes the set of neighbors of v in G: $N_G(v) = \{u \in V(G) \mid \{u, v\} \in E(G)\}$. The *degree* of v in G is $|N_G(v)|$. For $U \subseteq V(G)$, the *open neighborhood of* U *in* G, denoted by $N_G(U)$, is the set of vertices adjacent to some vertex in U but not belonging to U itself: $N_G(U) = (\bigcup_{v \in U} N_G(v)) \setminus U$. For an edge set $E \subseteq E(G)$ we denote by $V(E)$ the set of vertices of G incident to some edge in E.

We say that a vertex set $C \subseteq V(G)$ is *connected in* G if, for every $u, v \in C$, there is a walk of $G[C]$ starting with u and ending with v, where a *walk* of G is a sequence of vertices in which every vertex except for the last is adjacent to the next vertex in the sequence. It is a *connected component* or simply a *component* of G if it is connected and is inclusion-wise maximal subject to connectivity. For each vertex set S of G, we denote by $\mathcal{C}_G(S)$ the set of components of $G[V(G) \setminus S]$. When G is clear from the context we may drop the subscript and write $\mathcal{C}(S)$ for $\mathcal{C}_G(S)$. A vertex set $S \subseteq V(G)$ is a *separator* of G if $|\mathcal{C}_G(S)| \geq 2$. We usually assume that G is connected and therefore $\emptyset$ is not a separator of G. A component $C \in \mathcal{C}_G(S)$ is a *full component* associated with S if $N_G(C) = S$. A graph G is *biconnected* (*triconnected*) if every separator of G is of cardinality two (three) or greater. A graph is a *cycle* if it is connected and every vertex is adjacent to exactly two vertices. A graph is a *tree* if it is connected and does not contain a cycle as a subgraph. A tree is a *path* if it has exactly two vertices of degree one or its vertex set is a singleton. Those two vertices of degree one are called the *ends* of the path. If p is a path with ends a and b, we say that p is *between* a and b. When we speak of cycles or paths of G, we mean subgraphs of G that are cycles or paths. Let p be a cycle or path of G. A *chord* of p in G is an edge $\{u, v\}$ of G with $u, v \in V(p)$ that is not an edge of p. A cycle or path of G is *chordless* if it does not have any chord in G.

Tree decompositions. A *tree-decomposition* of G is a pair $(T, \mathcal{X})$ where T is a tree and $\mathcal{X}$ is a family $\{X_i\}_{i \in V(T)}$ of vertex sets of G, indexed by the nodes of T, such that the following three conditions are satisfied. We call each X_i the *bag* at node i.

1. $\bigcup_{i \in V(T)} X_i = V(G)$.
2. For each edge $\{u, v\} \in E(G)$, there is some $i \in V(T)$ such that $u, v \in X_i$.
3. For each $v \in V(G)$, the set of nodes $I_v = \{i \in V(T) \mid v \in X_i\} \subseteq V(T)$ is connected in T.

The *width* of this tree-decomposition is $\max_{i \in V(T)} |X_i| - 1$. The *treewidth* of G, denoted by $tw(G)$ is the smallest k such that there is a tree-decomposition of G of width k.

Minimal separators and potential maximal cliques. Let G be a graph and S a separator of G. For distinct vertices $a, b \in V(G)$, S is an *a-b separator* if there is no path between a and b in $G[V(G) \setminus S]$; it is a *minimal a-b separator* if it is an a-b separator and no proper subset of S is an a-b separator. A separator is a *minimal separator* if it is a minimal a-b separator for some $a, b \in V(G)$. A necessary and sufficient condition for a separator S of G to be minimal is that $\mathcal{C}_G(S)$ has at least two members that are full components associated with S. We say that a connected vertex set C of G is *minimally separated* if $N_G(C)$ is a minimal separator.

Graph H is *chordal* if every cycle of H with four or more vertices has a chord in H. H is a *triangulation of graph G* if it is chordal, $V(G) = V(H)$, and $E(G) \subseteq E(H)$. A triangulation H of G is *minimal* if there is no triangulation H' of G such that $E(H')$ is a proper subset of $E(H)$.

A vertex set X of G is a *potential maximal clique* (PMC for short) of G if there is some minimal triangulation H of G such that X is a maximal clique of H. We denote by $\Pi(G)$ the set of all PMCs of G. The following lemmas due to Bouchitté and Todinca [8] are essential in our reasoning about PMCs.

▶ **Lemma 1** (Bouchitté and Todinca [7]). *Let X be a PMC of a graph G. Then, for every component C of $G[V(G) \setminus X]$, $S = N_G[C]$ has a full component containing $X \setminus S$ and hence is a minimal separator. Moreover, for every minimal separator S of G such that $S \subseteq X$, we have $S \neq X$ and there is some component C of $G[V(G) \setminus X]$ such that $S = N_G(C)$.*

▶ **Lemma 2** (Bouchitté and Todinca [7]). *A vertex set X of graph G is a PMC of G if and only if all of the following conditions hold.*

1. *There is no full component associated with X.*
2. *For every $u, v \in X$, either $\{u, v\} \in E(G)$ or there is some $C \in \mathcal{C}_G(X)$ such that $u, v \in N_G(C)$.*

They also show that PMCs are extremely useful for computing the treewidth.

▶ **Theorem 3** (Bouchitté and Todinca [7]). *Given a graph G of n vertices and $\Pi(G)$, the treewidth of G can be computed in time $|\Pi(G)|n^{O(1)}$.*

We need the following property of PMCs.

▶ **Lemma 4** (Bouchitté and Todinca [7]). *Let X be a PMC of a graph G. Then, every proper subset of X has a full component associated with it. As a consequence, no proper subset of a PMC is a PMC.*

Plane graphs. A *sphere-embedded graph* is a graph where a vertex is a point on the sphere Σ and each edge is a simple curve on Σ between two distinct vertices that does not contain any vertex except for its ends. In the combinatorial view of a sphere-embedded graph G, an edge

represents the adjacency between two vertices at its ends. A sphere-embedded graph G is a *plane graph* if no two edges of G intersect each other as curves except at their ends. A graph G is *planar* if there is a plane graph that is isomorphic to G in the combinatorial view. We call this plane graph a *planar embedding* of G.

Let G be a biconnected plane graph. Then the removal of all the edges of G from the sphere Σ results in a collection of regions homeomorphic to open discs. We call these regions the *faces* of G. Each face of G is bounded by a simple closed curve that consists of edges of G. A vertex or an edge of G is *incident* to a face f of G, if it is contained in the closed curve that bounds f. We denote by $F(G)$ the set of faces of G. For each $f \in F(G)$, we denote by $V(f)$ the set of vertices of G incident to f.

A *combinatorial* representation of a plane graph G is an indexed set $\{\pi_v\}_{v\in V(G)}$, where π_v is a permutation on $N_G(v)$ such that $\pi_v(u)$ for $u \in N_G(v)$ is the neighbor of v that comes immediately after u in the clockwise order around v. Suppose we have two planar embeddings G_1 and G_2 of a graph G and assume that $V(G_1) = V(G_2) = V(G)$ and moreover the isomorphisms between G_1 and G as well as between G_2 and G are identities. Let, for $i \in \{1, 2\}$, the combinatorial representation of G_i be $\{\pi_{i,v}\}_{v\in V(G)}$. We say that these two planar embeddings are *combinatorially equivalent* if either $\pi_{1,v} = \pi_{2,v}$ for every $v \in V(G)$ or $\pi_{1,v} = (\pi_{2,v})^{-1}$ for every $v \in V(G)$. In this paper, we mainly deal with triconnected planar graphs. It is known that, for a triconnected planar graph G, the planar embedding of G is unique up to combinatorial equivalence [22]. For this reason, we will be dealing with triconnected plane graphs rather than triconnected planar graphs.

In conventional approaches for branch-decompositions and tree-decompositions of plane graphs [19, 6], the standard tools are radial graphs and nooses. We observe that, for triconnected plane graphs, latching graphs as defined below may replace those tools and greatly simplify definitions and reasoning.

▶ **Definition 5.** *Let G be a biconnected plane graph. The* latching graph *of G, denoted by L_G, is a multigraph obtained from G by, for each face f bounded by a cycle of four or more vertices, drawing every chord of the cycle within f.*

Figure 1 shows a triconnected plane graph with its latching graph and a plane graph for which the latching graph is not simple.

▶ **Proposition 6.** *The latching graph of a plane graph G is simple if G is triconnected.*

Proof. Let G be a triconnected plane graph and let u and v be two vertices of G. It is straightforward to see, and is observed in [6], that there are exactly two faces of G to which both u and v are incident if $\{u, v\}$ is an edge of G and there is at most one such face otherwise. In the first case, there is no face in which $\{u, v\}$ is a chord and, in the second case, there is at most one face in which $\{u, v\}$ is a chord. Thus, there is at most one edge between u and v in L_G and therefore L_G is simple. ◀

Although the latching graph L_G of a triconnected plane graph G is not a plane graph in general, we are interested in its subgraphs that are plane graphs. For $X \subseteq V(G)$, the *subgraph of L_G induced by X*, denoted by $L_G[X]$, is a sphere-embedded graph with vertex set X that inherits all edges of L_G, as curves, with two ends in X.

▶ **Proposition 7.** *Let G be a triconnected plane graph and let X be a vertex set of G. Then, $L_G[X]$ is a plane graph if and only if there is no face f of G such that $|V(f) \cap X| \geq 4$.*

Proof. Since $V(f) \cap X$ forms a clique of $L_G[X]$ for each face f of G, $L_G[X]$ is not a plane graph if there is some face f such that $|V(f) \cap X| \geq 4$. Conversely, if $L_G[X]$ has an edge-crossing, it must be in some face f of G and we have $|V(f) \cap X| \geq 4$. ◀

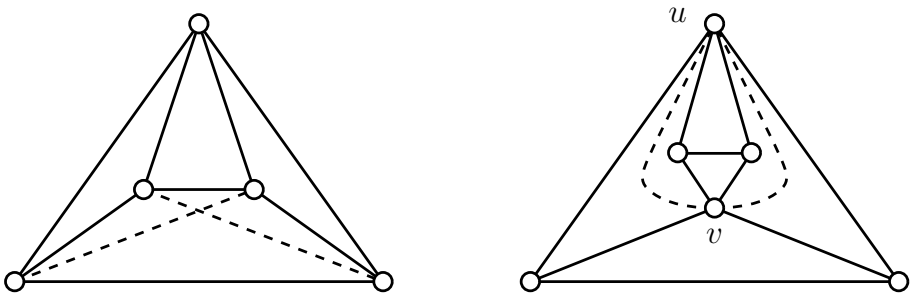


Figure 1 (A) a triconnected plane graph (solid edges) and its latching graph (solid and broken edges). (B) edge $\{u, v\}$ is drawn in more than one face if u and v separate the graph.

Let $X \subseteq V(G)$ such that $L_G[X]$ is a biconnected plane graph. We call each face of $L_G[X]$ a *region* of $L_G[X]$, in order to avoid confusions with the faces of G. We say that a region r of $L_G[X]$ is *empty* if r does not contain any vertex of G.

▶ **Proposition 8** (⋆). *Let G be a triconnected plane graph and let $X \subseteq V(G)$ be such that $L_G[X]$ is a biconnected plane graph. Then, each $C \in \mathcal{C}_G(X)$ is contained in some region of $L_G[X]$. Moreover, each region of $L_G[X]$ contains at most one component in $\mathcal{C}_G(X)$ and, if the region is incident to four or more vertices, exactly one component.*

We use this framework to formulate the fundamental observation on minimal separators of triconnected plane graphs, which is conventionally stated in terms of nooses [6].

▶ **Proposition 9** (⋆). *Let G be a triconnected plane graph. A vertex set S of G is a minimal separator of G if and only if $L_G[S]$ is a cycle and neither of the two regions of $L_G[S]$ is empty.*

▶ Remark 10. It follows from this proposition that every minimal separator of a triconnected planar graph has exactly two full components associated with it.

▶ **Corollary 11** (⋆). *Let G be a triconnected plane graph. A vertex set S of G with $|S| \geq 4$ is a minimal separator of G if and only if $L_G[S]$ is a cycle.*

The following fact is essential in our treatment of PMCs of triconnected plane graphs.

▶ **Proposition 12** (⋆). *For every PMC X of a triconnected plane graph G, $L_G[X]$ is a biconnected plane graph. Moreover, every region of $L_G[X]$ is bounded by a chordless cycle of L_G.*

3 Characterizing PMCs in triconnected planar graphs

In this section, we characterize PMCs of a triconnected plane graph in terms of graphs we call steerings.

▶ **Definition 13.** *Let γ be a cycle. A subset R of $V(\gamma)$ is a* slot *of γ if R is a singleton or an edge of γ. A graph H is a* steering*, if there is a bipartition (S, P) of $V(H)$ such that $H[S]$ is a cycle, $N_H(P)$ is neither empty nor a slot of $H[S]$, and if $|P| \geq 2$ then the following conditions hold.*

1. *$H[P]$ is a path.*
2. *No internal vertex of the path $H[P]$ is adjacent to any vertex in S.*
3. *For each end t of the path, $N_H(t) \cap S$ is a slot of $H[S]$.*

We call H an (S, P)-steering *in this situation. We call a steering a* wheel *if it is an (S, P)-steering for some bipartition (S, P) of $V(H)$ such that $|P| = 1$; otherwise it is a* non-wheel steering.

Figure 2 shows some steerings. Figure 3 shows two steerings with different (S, P)-bipartitions. The example (B) is an (S, P)-steering for the second bipartition but not for the first bipartition. Figure 4 shows some graphs that are not steerings.

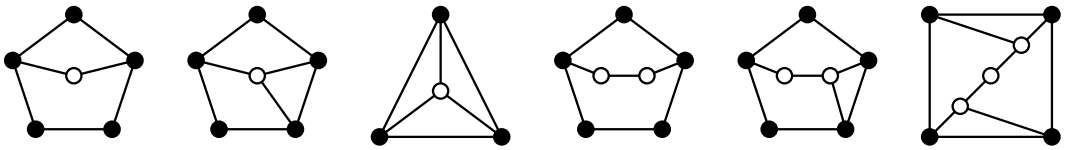
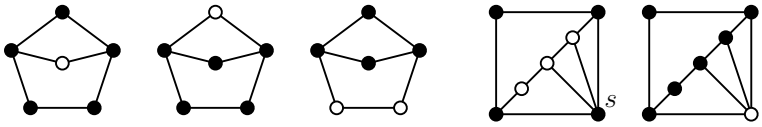


Figure 2 Some steerings. White vertices belong to P and black vertices belong to S, in a choice of the bipartition (S, P) that works. The first three are wheels while the remaining three are non-wheels.



(A) (B)

Figure 3 (A) A steering with three possible (S, P) bipartitions. Because of the first or the second bipartition, it is a wheel. (B) Also a wheel. For the first (S, P) bipartition, it is not an (S, P)-steering since an internal vertex of the path on P are adjacent to $s \in S$. The second (S, P) bipartition shows that it is a wheel.

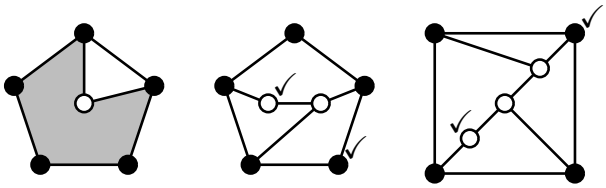


Figure 4 Some graphs that are not steerings. In the shown (S, P) bipartition, the first example H is not a steering since $N_H(P) \cap S$ is a slot. The second example is not a steering since $N_H(t) \cap S$ is not a slot for the right end t of the path $H[P]$. The third example is not a steering since an internal vertex of the path $H[P]$ is adjacent to a vertex in S. It can be verified that these graphs are not (S, P)-steerings for any other (S, P) bipartition. It is explained in the main text why any of these plane graphs cannot be $L_G[X]$ for a PMC X of a triconnected plane graph G.

▶ **Proposition 14** ($\star$). *Let H be a steering. Then H is biconnected and planar. Moreover, the planar embedding of H is unique up to combinatorial equivalence.*

From now on, when we refer to a steering H, we view H as a plane graph rather than a combinatorial graph as originally defined.

The following lemma shows that X is a PMC of a triconnected plane graph G if $L_G[X]$ is a steering. Before going into technical details, it may be helpful to study the examples in Figures 2 and 3 and confirm the following.

1. The cycle bounding each face is chordless.
2. Every pair of vertices is either adjacent to each other or incident to a common face.

Also observe that if $L_G[X]$ is one of the non-steering graphs in Figure 4, then X cannot be a PMC of G: in the first example, the component of $G[V(G) \setminus X]$ lying in the gray face would be a full component of X, violating the first condition of Lemma 2 for X to be a PMC of G; in the second and third examples, the pair of checked vertices do not share a common face and therefore cannot belong to the neighborhood of a common component of $G[V(G) \setminus X]$, violating the second condition of Lemma 2 for X to be a PMC of G.

▶ **Lemma 15.** *Let X be a vertex set of a triconnected plane graph G and suppose $L_G[X]$ is a steering. Then, X is a PMC of G.*

Proof. Suppose $L_G[X] = H$ where H is an (S, P)-steering. This equality means an equality as plane graphs, since we are viewing a steering as a biconnected plane graph, as we mentioned in the remark following Proposition 14.

Due to Proposition 8, each component in $\mathcal{C}_G(X)$ is contained in a region of H and each region of H contains at most one component in $\mathcal{C}_G(X)$.

We show that X satisfies the two conditions for PMCs in Lemma 2: (1) There is no full component associated with X; (2) For every pair of vertices x and y in X, either x is adjacent to y in G or there is some $C \in \mathcal{C}_G(X)$ such that $x, y \in N_G(C)$.

To show that Condition (1) holds, let C be an arbitrary component in $\mathcal{C}_G(X)$ and γ_C be the cycle bounding the region of H containing C. If γ_C is $H[S]$, then C is not a full component associated with X as S is a proper subset of X. Otherwise, there is a vertex in S that does not belong to γ_C, since $N_H(P)$ is not a slot of the cycle $H[S]$, and therefore C is not a full component associated with X.

To show that Condition (2) holds, we first show that, for arbitrary two members x and y of X, there is some region of H whose boundary contains both x and y. If $x, y \in S$, then this certainly holds since H has a region bounded by the cycle $H[S]$. So suppose that at least one of x and y, say x, belongs to P. We argue in a few cases.

First suppose that $|P| = 1$ and hence $P = \{x\}$ and $y \in S$. Then, there is certainly a region of H bounded by a cycle consisting of x and a subpath of the cycle $H[S]$ that contains y. See the first three examples in Figure 2.

Next suppose that $|P| \geq 2$. Since there is no internal vertex of the path $H[P]$ adjacent in H to any vertex in S, there are two regions of H bounded by cycles containing the path $H[P]$. Let γ_1 and γ_2 be those cycles. Then, γ_i, for each $i \in \{1, 2\}$, consists of $H[P]$ and a subpath p_i of $H[S]$. Since $N_H(t) \cap S$ for each end t of the path $H[P]$ is a slot, we have $V(p_1) \cap V(p_2) = S$. Therefore, either γ_1 or γ_2 contains both x and y. See the last three examples in Figure 2.

We are ready to show that Condition (2) holds. Let x and y be arbitrary two vertices in X. As we have shown above, there is a cycle γ bounding a region of H such that $x, y \in V(\gamma)$. If this region contains some $C \in \mathcal{C}_G(X)$, then we are done since $N_G(C) = V(\gamma)$. Otherwise, γ bounds an empty region and must be a triangle since otherwise $L_G[X]$ would not be a plane graph. Therefore, $\{x, y\}$ is an edge of $L_G[X]$. If this is an edge of G then we are done, so suppose not. Let γ' be another cycle bounding a region of $L_G[X]$ that contains the edge $\{x, y\}$. This region bounded by γ' cannot be empty since, if it were, then we would have a face of G incident to four or more vertices of X, and $L_G[X]$ would not be a plane graph. We are done, since $x, y \in N_G(C')$ where C' is the member of $\mathcal{C}_G(X)$ contained in the region bounded by γ'. ◀

The converse of this lemma is shown in several steps. We start with a technical proposition.

▶ **Proposition 16.** *Let X be a PMC of a triconnected plane graph G. If $L_G[X']$ is a steering for some $X' \subseteq X$, then $X = X'$ and hence $L_G[X]$ is a steering.*

Proof. If $L_G[X']$ is a steering, X' is a PMC of G due to Lemma 15. Since no proper subset of a PMC is a PMC (Lemma 4), we have $X = X'$. ◀

The next lemma deals with a special case where every minimal separator contained in a PMC X consists of three vertices.

▶ **Lemma 17.** *Let X be a PMC of a triconnected plane graph G and suppose that, for every $C \in \mathcal{C}_G(X)$, $|N_G(C)| = 3$. Then, $L_G[X]$ is a K_4.*

Proof. Recall that, due to Proposition 12, $L_G[X]$ is a biconnected plane graph. Recall also that every $C \in \mathcal{C}_G(X)$ is contained in a region of $L_G[X]$ bounded by a cycle whose vertex set is $N_G(C)$. Let γ be a cycle that bounds a region r of $L_G[X]$ and let $S = V(\gamma)$. If r is empty then $|S| = 3$ since otherwise $L_G[X]$ would not be a plane graph (Proposition 7). Otherwise r contains some $C \in \mathcal{C}_G(X)$ and we also have $|S| = 3$ due to the assumption. Let $x \in X \setminus S$ and $s \in S$ be arbitrary. We show that x is adjacent to s in $L_G[X]$. If x is adjacent to s in G then we are done. Otherwise, due to the second condition for PMCs in Lemma 2, there is some $C \in \mathcal{C}_G(X)$ such that $x, s \in N_G(C)$. Since the region of $L_G[X]$ containing C is bounded by a triangle on $N_G(C)$, x is adjacent to s in $L_G[X]$. Therefore, x is adjacent in $L_G[X]$ to every $s \in S$ and hence $L_G[S \cup \{x\}]$ is a K_4. As $L_G[S \cup \{x\}]$ is an $(S, \{x\})$-steering, we have $X = S \cup \{x\}$ due to Proposition 16 and hence $L_G[X]$ is a K_4. ◀

To deal with the more general case, we use the following notion of *arches*.

▶ **Definition 18.** *Let G be a triconnected plane graph and let S be a minimal separator of G with $|S| \geq 4$. An* arch *of S is a subset P of $V(G) \setminus S$ such that $L_G[P]$ is a path and $N_{L_G}(P) \cap S$ is neither empty nor a slot of the cycle $L_G[S]$.*

If $L_G[S \cup P]$ is an (S, P)-steering, then P is certainly an arch of S. The converse does not hold: if P is an arch of S then $L_G[S \cup P]$ is not necessarily an (S, P)-steering, as the definition of steerings requires more conditions to be satisfied by $L_G[S \cup P]$. We show, however, in the next lemma that if P is an inclusion-wise minimal arch then $L_G[S \cup P]$ is a steering. A caveat: $L_G[S \cup P]$ is not necessarily an (S, P)-steering in this case; it may be an $(S', \{s\})$-steering where $S' = (S \cup P) \setminus \{s\}$ for some $s \in S$. In Lemma 20, we show that if X is a PMC then every minimal separator $S \subset X$ with $|S| \geq 4$ has an arch P that is a subset of $X \setminus S$. It follows from these two lemmas that, for every minimal separator S with $|S| \geq 4$ that is a subset of a PMC X, there is some $P \subset X \setminus S$ such that $L_G[S \cup P]$ is a steering. Due to Lemma 15, $S \cup P$ is a PMC and, since no proper subset of a PMC is a PMC (Lemma 4), we conclude that $X = S \cup P$ and $L_G[X]$ is a PMC, which is an essential part of Theorem 21.

▶ **Lemma 19.** *Let S be a minimal separator of a triconnected plane graph G with $|S| \geq 4$. Suppose S has an arch and let P be an inclusion-wise minimal arch of S. Let $H = L_G[S \cup P]$. Then, either H is an (S, P)-steering or there is some $s \in N_{L_G}(P) \cap S$ such that H is an $((S \cup P) \setminus \{s\}, \{s\})$-steering.*

Proof. If there is some $v \in P$ such that $N_{L_G}(v) \cap S$ is neither empty nor a slot of the cycle $L_G[S]$, then $\{v\}$ is an arch of S and, since P is minimal, $P = \{v\}$. We are done, since H is an $(S, \{v\})$-steering.

Suppose $N_{L_G}(v) \cap S$ is either empty or a slot of $L_G[S]$ for every $v \in P$. Since $N_{L_G}(P) \cap S$ is not a slot of $L_G[S]$, there are two vertices $v_1, v_2 \in P$ and two vertices $s_1, s_2 \in S$ such that v_i is adjacent to s_i for $i \in \{1, 2\}$ and $\{s_1, s_2\}$ is not an edge of S. Let p be a shortest path in $L_G[P]$ between v_1 and v_2. Then, $V(p)$ is an arch of S and, since P is minimal, P must be equal to $V(p)$. If no internal vertex of p is adjacent in L_G to any vertex in S, then $L_G[S \cup P]$ is an (S, P)-steering and we are done. So suppose an internal vertex v of p is adjacent to some vertex in S. Let p_i be the subpath of p between v_i and v, for $i \in \{1, 2\}$. Since P is a minimal arch of S, neither $V(p_1)$ nor $V(p_2)$ is an arch of S. Therefore, $N_{L_G}(\{v_i, v\}) \cap S$ is a slot for $i \in \{1, 2\}$. This is possible only if there is a vertex $s \in S$ such that s is adjacent to both s_1 and s_2 on $L_G[S]$, $N_{L_G}(v_i)$ is either $\{s_i\}$ or $\{s_i, s\}$ for $i \in \{1, 2\}$, and $N_{L_G}(v) \cap S = \{s\}$. Moreover, since $P = V(p)$ is a minimal arch of S, no vertices in P other than v, v_1, or v_2 are adjacent to any vertex in S. Let p_3 be the subpath of $L_G[S]$ between s_1 and s_2 avoiding s and let γ be the cycle consisting of p_3 and p. Let $S' = V(\gamma) = (S \cup P) \setminus \{s\}$. We claim that γ is chordless in L_G, that is, $L_G[S'] = \gamma$. This is because p_3 is chordless since $L_G[S]$ is a cycle by assumption, p is chordless since it is the shortest path in P between v_1 and v_2, and the only edges between $V(p)$ and $V(p_3)$ are the edges $\{v_1, s_1\}$ and $\{v_2, s_2\}$. Since $N_{L_G}(s) \cap S' = \{s_1, v, s_2\}$, $L_G[S \cup P]$ is an $(S', \{s\})$-steering as desired. ◀

▶ **Lemma 20.** *Let X be a PMC of a triconnected plane graph G and let $S \subseteq X$ be a minimal separator of G such that $|S| \geq 4$. Then, there is an arch P of S such that $P \subseteq X \setminus S$.*

Proof. Due to Lemma 1, $X \setminus S$ is non-empty and is a subset of one of the two full components associated with S. Let r_0 be the region of $L_G[S]$ that contains $X \setminus S$.

Suppose that there is no arch P of S such that $P \subseteq X \setminus S$. Then, for every pair of vertices s and s' in S that are not adjacent to each other on the cycle $L_G[S]$, we can draw a curve between s and s' in r_0 without crossing the drawing of $L_G[X]$. Therefore, there is a region r of $L_G[X]$ contained in r_0 that is incident to all vertices in S. Let γ be the cycle of $L_G[X]$ bounding r. Since $|V(\gamma)| \geq |S| \geq 4$, r is non-empty (Proposition 8) and hence contains a component $C \in \mathcal{C}_G(X)$. Therefore, $V(\gamma)$ must be a minimal separator of G, due to Lemma 1. But this is impossible, since some edge of $L_G[S]$ is a chord of γ, contradicting Proposition 9. ◀

The following is our main theorem in this section.

▶ **Theorem 21.** *A vertex set X of a triconnected plane graph G is a PMC if and only if $L_G[X]$ is a steering.*

Proof. Lemma 15 shows that if $L_G[X]$ is a steering then X is a PMC. We prove the other direction. Let X be a PMC of G.

If $|N_G(C)| = 3$ for every $C \in \mathcal{C}_G(X)$, $L_G[X]$ is a K_4 due to Lemma 17 and hence is an $(X \setminus \{x\}, \{x\})$-steering for every $x \in X$.

Suppose there is some $C \in \mathcal{C}_G(X)$ with $|S| \geq 4$ where $S = N_G(C)$. Due to Lemma 20, there is an arch of S that is a subset of $X \setminus S$. Let P be an inclusion-wise minimal arch of S that is a subset of $X \setminus S$. Due to Lemma 19, $L_G[S \cup P]$ is a steering and hence $S \cup P$ is a PMC of G, due to Lemma 15. Due to Proposition 16, $X = S \cup P$ and hence $L_G[X]$ is a steering. ◀

4 Polynomial delay generation of PMCs of a triconnected planar graph

Let J be some combinatorial structure of size n and let $\mathcal{S}(J)$ be the set of some combinatorial objects defined on J. We say an algorithm *generates* $\mathcal{S}(J)$ *with polynomial delay* if it outputs each member of $\mathcal{S}(J)$ exactly once and the time between two consecutive events is $n^{O(1)}$, where an event is the initiation of the algorithm, an output, or the termination of the algorithm.

The basic approach for generating $\Pi(G)$ for a given triconnected plane graph G is as follows. PMCs X such that $|N_G(C)| = 3$ for every $C \in \mathcal{C}_G(X)$ can be trivially generated with polynomial delay, since $L_G[X]$ is a K_4 for each such X due to Lemma 17. So, we concentrate on generating $\Pi'(G)$ consisting of members X of $\Pi(G)$ such that $\mathcal{C}_G(X)$ contains a component C with $|S| \geq 4$ where $S = N_G(C)$.

We need an algorithm to generate minimal separators of G. Although a polynomial delay algorithm for this task is known for general graphs [2], we use an algorithm specialized for triconnected plane graphs for a technical reason to become clear below.

We need the following result due to Uno and Satoh [21].

▶ **Lemma 22** (Uno and Satoh [21])**.** *Given a graph G, all chordless cycles of G can be generated with polynomial delay. Given a graph G and two vertices $s, t \in V(G)$, all chordless paths of G between s and t can be generated with polynomial delay.*

▶ **Lemma 23.** *Given a triconnected plane graph G, all minimal separators of G can be generated with polynomial delay. Moreover, for each $v \in V(G)$, all minimal separators of G that do not contain v can be generated with polynomial delay.*

Proof. Given G, we generate all chordless cycles of L_G with polynomial delay, using the algorithm of Uno and Satoh. These chordless cycles are in one-to-one correspondence with minimal separators of G due to Proposition 9. For the second part, we generate all chordless cycles of $G[V(G) \setminus \{v\}]$. Since a chordless cycle of $G[V(G) \setminus \{v\}]$ is a chordless cycle of G, we have the result. ◀

▶ Remark 24. It is not clear if the algorithm [2] for generating minimal separators of a general graph can be adapted to support the second part of the lemma, which is used in the proof of Theorem 32. We also note that the chordless path part of Lemma 22 is used in the proof of Lemma 30.

To design an algorithm to generate all PMCs based on our characterization of PMCs stated in Theorem 21, we need some preparations.

▶ **Definition 25.** *Let G be a triconnected plane graph and C a minimally separated component of G with $|S| \geq 4$ where $S = N_G(C)$. Recall that there are exactly two full components associated with S and let C' be the full component distinct from C. A* port *of (S, C) is a vertex $u \in C'$ such that $N_{L_G}(u) \cap S$ is a slot of $L_G[S]$. We call the slot $N_{L_G}(u) \cap S$ of $L_G[S]$ the* slot for u*. A pair of ports u_1 and u_2 is* valid *if the union of the slot for u_1 and the slot for u_2 is not a slot. A vertex $s \in S$ with two neighbors s_1 and s_2 on the cycle $L_G[S]$ is a* hinge *of a valid pair of ports u_1 and u_2 if the slot for u_1 is either $\{s_1\}$ or $\{s_1, s\}$ and the slot for u_2 is either $\{s_2\}$ or $\{s_2, s\}$. See Figure 5 for examples of ports, valid and invalid port pairs, and a hinge.*

Ports are potential ends of a path p such that $L_G[X]$ for $X = S \cup V(p)$ is a steering. A pair of ports must be valid in order for the pair to be the ends of such path p. We generate such paths using the algorithm for generating chordless paths. The following lemma is the basis of this approach.

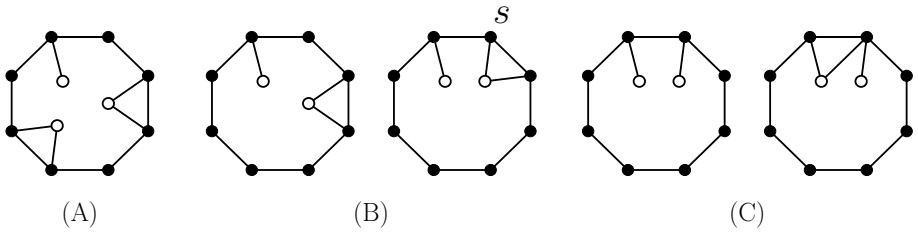


Figure 5 (A) White vertices are some ports of (S, C) where S consists of black vertices and C is the full component associated with S that lies in the outer face of the black cycle. (B) Some valid pairs of ports. The valid pair in the second example has a hinge s. (C) Some invalid pairs of ports.

▶ **Lemma 26.** *Let G be a triconnected plane graph and C a minimally separated component of G with $|S| \geq 4$, where $S = N_G(C)$. Let C' be the other full component of S. Let P be an arbitrary subset of C' with $|P| \geq 2$ and let $X = S \cup P$.*

1. *For each valid pair u_1 and u_2 of ports of (S, C), let $A(C, u_1, u_2)$ be the subgraph of L_G induced by the vertex set $(C' \setminus N_{L_G}(S)) \cup \{u_1, u_2\}$. Then, $L_G[X]$ is an (S, P)-steering if and only if there is a valid pair u_1 and u_2 of the ports of S such that $P = V(p)$ for some chordless path p of $A(C, u_1, u_2)$ between u_1 and u_2.*
2. *For each valid pair u_1 and u_2 of ports of (S, C) that has hinges and each hinge s of the pair, let $B(C, u_1, u_2, s)$ be the subgraph of L_G induced by the vertex set $(C' \setminus N_{L_G}(S)) \cup N_{L_G}(s) \cup \{u_1, u_2\}$. Then, $L_G[X]$ is an $(S', \{s\})$-steering for $s \in S$, where $S' = X \setminus \{s\}$, if and only if there is a valid pair of ports u_1 and u_2 of (S, C) with a hinge s such that $P = V(p)$ for some chordless path p of $B(C, u_1, u_2, s)$ between u_1 and u_2.*

Proof. For part 1, suppose $L_G[X]$ is an (S, P)-steering where $P \subseteq C'$. Let $H = L_G[X]$. Due to Definition 13, $H[P] = L_G[P]$ is a path. Let u_1 and u_2 be the two ends of this path. Then, u_1 and u_2 are ports of S and, moreover, the pair of these ports is valid. Since no internal vertex of the path $L_G[P]$ is adjacent in L_G to any vertex in S, $L_G[P]$ is a path in $A(C, u_1, u_2)$. It is a chordless path, since if it had a chord in $A(C, u_1, u_2)$, then the graph $L_G[P]$ would contain that chord and $L_G[P]$ would not be a path.

For the other direction, suppose that there is a valid pair of ports u_1 and u_2 of S such that $L_G[P]$ is a chordless path of $A(C, u_1, u_2)$ between u_1 and u_2. Then, $N_{L_G}(P \setminus \{u_1, u_2\}) \cap S$ is empty and hence $L_G[S \cup P]$ is an (S, P)-steering.

For part 2, suppose $L_G[X]$ is an $(S', \{s\})$-steering where $s \in S$ and $S' = X \setminus \{s\}$. Due to the proof of Lemma 19, $L_G[P]$ is a path of L_G between some valid pair of ports u_1 and u_2 of (S, C) and, moreover, s is a hinge of this pair of ports. Since the only vertex in S that can be adjacent to any internal vertex of $L_G[P]$ is s, $L_G[P]$ is a path in $B(C, u_1, u_2, s)$ between u_1 and u_2. It is a chordless path, since if it had a chord in $B(C, u_1, u_2, s)$, then the graph $L_G[P]$ would contain that chord and $L_G[P]$ would not be a path.

For the other direction, suppose that there is a valid pair of ports u_1 and u_2 of S with a hinge s such that $L_G[P]$ is a chordless path of $B(C, u_1, u_2, s)$ between u_1 and u_2. Then, $L_G[S']$, where $S' = X \setminus \{s\}$, is a cycle and $L_G(X)$ is an $(S', \{s\})$-steering. ◀

In implementing the generation of PMCs based on this lemma, we need to deal with the problem of suppressing duplicate outputs of a single element without introducing super-polynomial delay. We use the technique due to Bergougnoux, Kanté and Wasa [1] to address this problem. They used this technique for a particular problem of generating what they call minimal disjunctive separators. We formulate their technique in the following theorem that can be used for wide range of generation problems.

Algorithm 1 Generation of $\mathcal{S}(G)$, using sub-generators $\textsc{Gen}_i$ for $\mathcal{S}_i(G)$, $i \in I$.

▷ We say *emit* s to mean that this algorithm outputs s, in order to distinguish this action from the output events of the sub-generators
▷ We say that an output of a sub-generator is *suppressed* if it is not emitted

Ensure: Generate all members of $\mathcal{S}(G)$

for each $i \in I$ **do**
  Initiate $\textsc{Gen}_i$ and immediately suspend it
end for
$i^* \leftarrow 1$
repeat
  $ascending \leftarrow true$
  while $ascending$ **do**
    Execute $\textsc{Gen}_{i^*}$ up to the next event
    if the event is the termination event **then**
      $ascending \leftarrow false$
    else ▷ the event is an output event
      Let s be the object to be output
      if $s \in \mathcal{S}_i(G)$ for some $i > i^*$ **then** ▷ the output of s is suppressed
        $i^* \leftarrow$ the smallest $j > i^*$ such that $\textsc{Gen}_j$ has not been terminated
        ▷ such j exists, see the proof
      else
        Emit s
        $ascending \leftarrow false$
      end if
    end if
  end while
  if there is some j such that $\textsc{Gen}_j$ has not been terminated **then**
    $i^* \leftarrow$ the smallest j such that $\textsc{Gen}_j$ has not been terminated
  end if
until $\textsc{Gen}_i$ has been terminated for every $i \in I$

Let G be a graph on n vertices. We consider the general task of generating some structures defined on G. Let $\mathcal{S}(G)$ denote the set of those structures to be generated. Suppose N subsets $\mathcal{S}_1(G), \ldots, \mathcal{S}_N(G)$ of $\mathcal{S}(G)$ are defined. Let $I = \{i \mid 1 \leq i \leq N\}$. Suppose that the following conditions are satisfied.

1. $\mathcal{S}(G) = \bigcup_{i \in I} \mathcal{S}_i(G)$.
2. $N = n^{O(1)}$.
3. For each $s \in \mathcal{S}(G)$ and $i \in I$, it can be decided whether $s \in \mathcal{S}_i(G)$ in time $n^{O(1)}$.
4. For each $i \in I$, there is an algorithm $\textsc{Gen}_i$ that generates $\mathcal{S}_i(G)$ with polynomial delay.

▶ **Theorem 27.** *Under the above assumptions, $\mathcal{S}(G)$ can be generated with polynomial delay.*

We prove this theorem by showing that Algorithm 1 generates $\mathcal{S}(G)$ with polynomial delay. The idea of this algorithm is as follows. For each $s \in \mathcal{S}(G)$, let us say that $i \in I$ *owns* s if i is the largest j such that $s \in \mathcal{S}_j(G)$. We let $\textsc{Gen}_i$, where i owns s, be responsible for the output of s and suppress the output of s from $\textsc{Gen}_j$ for other j. The executions of $\textsc{Gen}_i$ for $i \in I$ are scheduled in such a way that at most a polynomial number of outputs are suppressed before an unsuppressed output occurs.

▶ **Lemma 28** (⋆). *Algorithm 1 generates $\mathcal{S}(G)$ with polynomial delay, if all of the four conditions above are satisfied.*

We now turn to the application of this technique to our goal.

For a minimally separated component C of a triconnected plane graph G with $|S| \geq 4$ where $S = N_G(C)$, let $\Pi(G, C)$ denote the set of PMCs X such that $C \in \mathcal{C}_G(X)$ and $L_G[X]$ is a steering.

▶ **Proposition 29.** *Let G be a triconnected plane graph and C a minimally separated component of G such that $|N_G(C)| \geq 4$. Then, $\Pi(G, C) \neq \emptyset$.*

Proof. Let $S = N_G(C)$ and let C' be the other full component of S. Let H be an arbitrary minimal triangulation of G in which S is a clique. Let X be an arbitrary maximal clique of $H[S \cup C']$ that contains S. Since S is a separator of H, X is a maximal clique of H. Since no maximal clique of a chordal graph is a minimal separator, X is a proper superset of S and belongs to $\Pi(G, C)$. ◀

▶ **Lemma 30.** *Given a triconnected plane graph G and a minimally separated component C of G with $|S| \geq 4$ where $S = N_G(C)$, $\Pi(G, C)$ can be generated with polynomial delay.*

Proof. Based on Lemma 26, we generate $\Pi(G, C)$ as follows. We first generate $X \in \Pi(G, C)$ such that $L_G[X]$ is an (S, P)-steering where $P = X \setminus S$. To do so, for every valid pair of ports u_1 and u_2 of S, we generate all chordless paths of $A(C, u_1, u_2)$ between u_1 and u_2, where $A(C, u_1, u_2)$ is as defined in Lemma 26, using the algorithm due to Sato and Uno given in Lemma 22. For each path p generated, we output $S \cup V(p)$. Since the number of valid pairs of ports is $O(n^2)$, the generation of X in this category is with polynomial delay even if some valid pairs do not yield any such X.

We then generate $X \in \Pi(G, C)$ such that $L_G[X]$ is an $(S', \{s\})$-steering where $s \in S$ and $S' = X \setminus \{s\}$. To do so, for every valid pair of ports u_1 and u_2 of S with hinges and for each hinge s of this pair, we generate all chordless paths of $B(C, u_1, u_2, s)$ between u_1 and u_2. For each path p generated, we output $S \cup V(p)$. The valid pair of u_1 and u_2 may have more than one ports when $|S| = 4$, which could cause duplicate outputs of an identical X. We suppress such duplicate outputs using Theorem 27. Let $\mathcal{T}$ be the set of all triples (u_1, u_2, s) where (u_1, u_2) is a valid pair of ports of (S, C) and s is a hinge of this pair. Index the set $\mathcal{T}$ by $I = \{1, \ldots, N\}$ where $N = |\mathcal{T}|$. For each $i \in I$, let $\mathcal{P}_i$ denote the set of chordless paths of $B(C, u_1, u_2, s)$ between u_1 and u_2, where (u_1, u_2, s) is the triple in $\mathcal{T}$ indexed by i. Let $\mathcal{P} = \bigcup_{i \in I} \mathcal{P}_i$. Our goal is to generate $\mathcal{P}$ without duplication. We verify the conditions for Theorem 27 to be applied.

1. $\mathcal{P} = \bigcup_{i \in I} \mathcal{P}_i$ by definition.
2. $N = |I| = O(n^3)$.
3. For each path p in $\mathcal{P}$ and each $i \in I$, it can be decided whether $p \in \mathcal{P}_i$ in time $n^{O(1)}$.
4. For each $i \in I$, we can generate $\mathcal{P}_i$ with polynomial delay as described above.

Therefore, Theorem 27 applies and we can generate $\mathcal{P}$ and hence all PMCs X in this category, with polynomial delay. ◀

▶ Remark 31. Since the PMCs X in the second category in the above lemma, those such that $L_G[X]$ is an $(S', \{s\})$-steering for some $s \in S$ where $S' = X \setminus \{s\}$, also belong to $\Pi(G, C')$ for some full component associated with S', it might appear that the cumbersome generation of those PMCs described in the above proof is unnecessary. However, if we generate only the members of $\Pi(G, C)$ in the first category, those X such that $L_G[X]$ is an (S, P)-steering where $P = X \setminus S$, then it is possible for the generator of $\Pi(G, C)$ to produce no outputs. This is a problem, since there is no readily provable polynomial bound on the number of successive components C for which the generation of $\Pi(G, C)$ produces no outputs.

The following is the main result of this section.

▶ **Theorem 32.** *Given a triconnected plane graph G, $\Pi(G)$ can be generated with polynomial delay.*

Proof. It suffices to show that $\Pi'(G)$, the set of PMCs X of G such that $|N_G(C)| \geq 4$ for some $C \in \mathcal{C}_G(X)$, can be generated with polynomial delay. Let $n = |V(G)|$. For each $v \in G$, let $\Pi_v(G)$ denote the union of $\Pi(G, C)$ over all minimally separated components C of G such that $v \in C$ and $|N_G(C)| \geq 4$. We have $\Pi'(G) = \bigcup_{v \in V(G)} \Pi_v(G)$ and $|V(G)| = n = n^{O(1)}$. Given $X \in \Pi(G)$ and $v \in V(G)$, it is straightforward to test if $X \in \Pi_v(G)$ in polynomial time. For each v, $\Pi_v(G)$ can be generated with polynomial delay: we generate all minimal separators S such that $v \notin S$ by the second part of Lemma 23 and, for each S generated such that $|S| \geq 4$, generate $\Pi(G, C)$ for the full component C of S such that $v \in C$. The delay in this algorithm is polynomial, since minimal separators are generated with polynomial delay and, for each minimal separator generated, at least one PMC is generated due to Proposition 29.

Therefore, applying Theorem 27, we can generate $\Pi(G)$ with polynomial delay. ◀

5 Computing the planar treewidth

An algorithm for generating PMCs immediately leads to a treewidth algorithm for triconnected planar graphs because of Theorem 3 due to Bouchitté and Todinca [7]. To extend the algorithm for general planar graphs, we use the following result from [5]. A vertex set S of a graph G is an *almost clique* of G if there is some $v \in S$ such that $S \setminus \{v\}$ is a clique of G.

▶ **Theorem 33** (Bodlaender and Koster [5])**.** *Let G be a graph and S a minimal separator of G that is an almost clique. Let C_i, $1 \leq i \leq m$, be the components of $G[V(G) \setminus S]$ and let $G_i = G[C_i \cup N_G(C_i)] \cup K(N_G(C_i))$, for $i = 1, \ldots, m$, be the graph obtained from $G[C_i \cup N_G(C_i)]$, the subgraph of G induced by the closed neighborhood of C_i, by filling the open neighborhood of C_i into a clique. Then $tw(G) = \max\{|S|, \max_{i \in \{1,\ldots,m\}} tw(G_i)\}$.*

We use this theorem for the special case where S is a two-vertex minimal separator: such S is always an almost clique.

▶ **Theorem 34.** *Given a planar graph G, $tw(G)$ can be computed in time $|\Pi(G)|n^{O(1)}$.*

Proof. We assume that G is biconnected. Otherwise, we compute the treewidth of each biconnected component and take the maximum.

We first prove the case where G is triconnected. We compute a planar embedding of G in time $O(n)$ [15]. Then we apply our generation algorithm to compute $\Pi(G)$ in time $|\Pi(G)|n^{O(1)}$. Finally, we apply Theorem 3 to obtain $tw(G)$ in time $|\Pi(G)|n^{O(1)}$.

For a general planar graph G, let $s(G)$ denote the number of two-vertex separators of G. We prove the theorem by induction on $s(G)$. The base case $s(G) = 0$, where G is triconnected, is already proved. Suppose $s(G) > 0$ and let S be an arbitrary two-vertex separator of G. Let $C_1, \ldots, C_m$ be the components of $G[V(G) \setminus S]$. Since G is biconnected, we have $N(C_i) = S$ for every $i \in \{1, \ldots, m\}$. Therefore, S is a minimal separator of G and, due to Theorem 33, we have $tw(G) = \max\{2, \max_{i \in \{1,\ldots,m\}} tw(G_i)\}$ where $G_i = G[C_i \cup S] \cup K(S)$. Since G_i is a subgraph of a minor of G obtained by contracting $C_{i'} \cup v$ into a single vertex for arbitrary $i' \neq i$ and $v \in S$, G_i is planar. Moreover, G_i is biconnected since any cut vertex of G_i would be a cut vertex of G. Therefore, we may apply the induction hypothesis to each G_i and compute $tw(G_i)$ for each i in time $|\Pi(G_i)|n^{O(1)}$. We claim that $\sum_i |\Pi(G_i)| \leq |\Pi(G)|$. To see

this, let X be an arbitrary PMC of G_i and H a minimal triangulation of G_i in which X is a maximal clique. Let H' be an arbitrary minimal triangulation of G such that $H'[V(G_i)] = H$. Since every minimal separator of G that is a clique in a minimal triangulation of G is a minimal separator of that triangulation [14], S is a minimal separator of H'. Therefore, X is a maximal clique of H' and hence is a PMC of G. Moreover, X cannot be a PMC of G_j for any $j \neq i$ since X is not a subset of $V(G_j)$. Therefore, the claim holds. We conclude that we can compute $tw(G)$ in time $|\Pi(G)|n^{O(1)}$. ◀

6 Future work

Our success in a special case of the open question whether $\Pi(G)$ can be computed in time $|\Pi(G)|n^{O(1)}$ does not seem to suggest any approach to solving the question for general graphs. Much more modest goal is to address the question for general planar graphs. It might be possible to extend our generation algorithm to general planar graphs by finding some way of handling PMCs that cross two-vertex minimal separators. Bart Jansen asked if the generation result could be extended to graphs embedded on tori. We observe that latching graphs are not appropriate tool for that purpose, since if G is a triconnected graph embedded on a torus and X is a PMC of G then the subgraph of the latching graph of G, appropriately defined on tori, induced by X may have edge crossings: the proof of Proposition 12 relies essentially on the assumption that G is embedded in the sphere. Studying upper bounds on $|\Pi(G)|$ is another avenue of research. There is a class of planar graphs for which $|\Pi(G)| = \Omega(1.442^n)$ [9]. These graphs are biconnected but not triconnected, so it would be interesting to ask if an upper bound smaller than this bound holds for triconnected planar graphs.

References

1 Benjamin Bergougnoux, Mamadou Moustapha Kanté, and Kunihiro Wasa. Disjunctive minimal separators enumeration. In *International Workshop on Enumeration Problems and Applications*, 2019. URL: `https://benjaminbergougnoux.github.io/pdf/bkw19.pdf`.

2 Anne Berry, Jean Paul Bordat, and Olivier Cogis. Generating all the minimal separators of a graph. *Int. J. Found. Comput. Sci.*, 11(3):397–403, 2000. `doi:10.1142/S0129054100000211`.

3 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, December 1996. `doi:10.1137/S0097539793251219`.

4 Hans L Bodlaender, Leizhen Cai, Jianer Chen, Michael R Fellows, Jan Arne Telle, and Dániel Marx. Open problems in parameterized and exact computation—IWPEC 2006. *Department of Information and Computing Sciences, Utrecht University*, 2006.

5 Hans L. Bodlaender and Arie M. C. A. Koster. Safe separators for treewidth. *Discret. Math.*, 306(3):337–350, 2006. `doi:10.1016/J.DISC.2005.12.017`.

6 Vincent Bouchitté, Frédéric Mazoit, and Ioan Todinca. Chordal embeddings of planar graphs. *Discret. Math.*, 273(1-3):85–102, 2003. `doi:10.1016/S0012-365X(03)00230-9`.

7 Vincent Bouchitté and Ioan Todinca. Treewidth and minimum fill-in: Grouping the minimal separators. *SIAM J. Comput.*, 31(1):212–232, 2001. `doi:10.1137/S0097539799359683`.

8 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`.

9 Fedor V. Fomin, Dieter Kratsch, Ioan Todinca, and Yngve Villanger. Exact algorithms for treewidth and minimum fill-in. *SIAM J. Comput.*, 38(3):1058–1079, 2008. `doi:10.1137/050643350`.

10 Fedor V. Fomin, Ioan Todinca, and Yngve Villanger. Exact algorithm for the maximum induced planar subgraph problem. In Camil Demetrescu and Magnús M. Halldórsson, editors, *Proceedings of the 19th Annual European Symposium, ESA 2011*, volume 6942 of *Lecture Notes in Computer Science*, pages 287–298. Springer, 2011. `doi:10.1007/978-3-642-23719-5_25`.

11 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.

12 Fedor V. Fomin and Yngve Villanger. Finding induced subgraphs via minimal triangulations. In Jean-Yves Marion and Thomas Schwentick, editors, *27th International Symposium on Theoretical Aspects of Computer Science, STACS 2010, March 4-6, 2010, Nancy, France*, volume 5 of *LIPIcs*, pages 383–394. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2010. doi:10.4230/LIPICS.STACS.2010.2470.

13 Fedor V. Fomin and Yngve Villanger. Treewidth computation and extremal combinatorics. *Comb.*, 32(3):289–308, 2012. doi:10.1007/S00493-012-2536-Z.

14 Pinar Heggernes. Minimal triangulations of graphs: A survey. *Discrete Mathematics*, 306(3): 297–317, 2006. doi:10.1016/J.DISC.2005.12.003.

15 John E. Hopcroft and Robert Endre Tarjan. Efficient planarity testing. *J. ACM*, 21(4):549–568, 1974. doi:10.1145/321850.321852.

16 Torsten Inkmann. *Tree-based decompositions of graphs on surfaces and applications to the Traveling Salesman Problem.* PhD thesis, Georgia Institute of Technology, 2008. URL: http://hdl.handle.net/1853/22583.

17 Frank Kammer and Torsten Tholey. Approximate tree decompositions of planar graphs in linear time. *Theoretical Computer Science*, 645:60–90, 2016. doi:10.1016/j.tcs.2016.06.040.

18 Tuukka Korhonen and Daniel Lokshtanov. An improved parameterized algorithm for treewidth. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023*, pages 528–541. ACM, 2023. doi:10.1145/3564246.3585245.

19 Paul D. Seymour and Robin Thomas. Call routing and the ratcatcher. *Comb.*, 14(2):217–241, 1994. doi:10.1007/BF01215352.

20 Hisao Tamaki. Positive-instance driven dynamic programming for treewidth. *J. Comb. Optim.*, 37(4):1283–1311, 2019. doi:10.1007/S10878-018-0353-Z.

21 Takeaki Uno and Hiroko Satoh. An efficient algorithm for enumerating chordless cycles and chordless paths. In Saso Dzeroski, Pance Panov, Dragi Kocev, and Ljupco Todorovski, editors, *Proceedings of the 17th International Conference of Discovery Science, DS 2014*, volume 8777 of *Lecture Notes in Computer Science*, pages 313–324. Springer, 2014. doi:10.1007/978-3-319-11812-3_27.

22 Hassler Whitney. 2-isomorphic graphs. *American Journal of Mathematics*, 55(1):245–254, 1933. doi:10.2307/2371127.

Efficient Enumeration of k-Plexes and k-Defective Cliques

Mohamed Jiddou
LI-PARAD, Université de Versailles Saint-Quentin-En-Yvelines, Paris-Saclay, France

George Manoussakis
LI-PARAD, Université de Versailles Saint-Quentin-En-Yvelines, Paris-Saclay, France

Abstract

We investigate the enumeration of dense subgraphs under two well-known relaxations of cliques: k-plexes and k-defective cliques. Our main contribution is a family of algorithms with improved worst-case and output-sensitive complexities, driven by a decomposition technique based on graph degeneracy. We first propose a worst-case output-size near-optimal algorithm to enumerate all maximal k-plexes of size at least $2k-1$, achieving a total time complexity of $\mathcal{O}(n(dk)^3 2^d \Delta^k)$, where d is the degeneracy and Δ the maximum degree of the input graph. We then refine this result to obtain a fixed-parameter tractable output-sensitive algorithm with complexity $\mathcal{O}(\alpha f(k)p(d\Delta))$, where α is the number of solutions, $f(k)$ is an arbitrary function of k, and p is a polynomial. We then extend this framework to the enumeration of k-defective cliques and also show a linear-time $O(n)$ algorithm for the enumeration of 2-plexes for graphs with bounded degeneracy. To the best of our knowledge, these complexities are competitive with or better than the current state of the art.

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

Keywords and phrases Parameterized complexity, enumeration algorithms, maximal cliques enumeration

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.22

1 Introduction

Finding dense substructures in graphs is a fundamental problem in data mining [9, 19], social network analysis [16, 23], bio-informatics [15, 14], and database theory, where uncovering dense groups of entities reveals critical insights into the organization of complex systems. In data mining, dense patterns often signify meaningful associations or frequent co-occurrences; in social networks, they correspond to densely connected communities; in biological networks, they may represent protein complexes or functional modules; and in database systems, they inform efficient indexing and query optimization over graph-structured data.

Among the most well-known dense patterns is the clique, a subset of vertices in which every pair is mutually adjacent. Cliques represent idealized notions of cohesion but are rarely found in real-world networks, which are typically large, sparse, and noisy. The requirement of full connectivity makes cliques overly restrictive for practical applications. As a result, various relaxed clique models have been proposed to capture dense – but not necessarily complete – subgraphs that better reflect real-world connectivity.

Among these, the k-plex and k-defective clique models have been widely used [2, 19, 10, 20, 24]. A k-plex allows each vertex to be non-adjacent to up to k other vertices in the subgraph, while a k-defective clique requires the induced subgraph to miss at most k edges. These definitions offer a balance between density and noise tolerance, enabling the discovery of dense structures even in imperfect data. These models have demonstrated practical utility in numerous domains: detecting non-clique-like but cohesive communities in social networks [20, 23], identifying biologically relevant modules in protein-protein interaction networks [22], and mining patterns in incomplete or uncertain data sources [8].

© Mohamed Jiddou and George Manoussakis;
licensed under Creative Commons License CC-BY 4.0

The versatility of relaxed clique models makes them an important tool in modern graph analysis, where the goal is often to find structures that are both dense and resilient to noise. Consequently, the development of efficient algorithms to enumerate such subgraphs has become a central research problem.

2 State of the art

In this paper, we focus on the enumeration of two families of graph-theoretic structures: k-defective cliques and k-plexes. Both serve as models of relaxed cliques. Although the enumeration of k-plexes has been extensively studied, less attention has been paid to k-defective cliques. In this section, we review existing techniques and results concerning the enumeration of maximal k-plexes and k-defective cliques. In both the related work and throughout this paper, we encounter two common classes of output-sensitive algorithms for enumeration problems: *Polynomial Total Time* and *Polynomial Delay*. A problem is said to be solvable in polynomial total time if the total time required to enumerate all solutions is bounded by a polynomial in the size of the input and the number of solutions. This notion is typically used when the user is interested in generating the full output. However, Polynomial Delay ensures that the time between any two consecutive outputs is bounded by a polynomial in the input size. This is particularly valuable in practice, as it allows the user to process or visualize each solution incrementally without waiting for the enumeration to complete.

To our knowledge, there exist only two enumeration algorithms specifically designed for maximal s-defective cliques, which are given in [10]. The first is a polynomial delay algorithm, meaning that the time elapsed between two consecutive outputs is bounded by a polynomial function. Specifically, the delay is $\mathcal{O}(n^2 N_m^{2k+1}/4^k)$, where n is the number of vertices in the graph, N_m is the size of the largest k-defective clique. The second algorithm, which follows a more brute force approach and whose complexity is not output sensitive, uses a branch-and-bound approach with an optimized pivoting technique. Its time complexity is $\mathcal{O}(n\alpha_s^n)$, where α_s is a real constant strictly less than 2. An optimization based on the degeneracy ordering of the graph reduces this complexity to $\mathcal{O}(n^{k+2}\alpha_k^\delta)$, where δ is the degeneracy of the graph, a parameter that is typically small in real-world networks.

For the problem of enumerating all maximal k-plexes, several algorithms have been proposed. Some of these are adaptations of the Bron–Kerbosch algorithm [3], which was originally designed to enumerate maximal cliques. Wu and Pei [20] extended the Bron–Kerbosch algorithm to handle maximal k-plexes by introducing a set of pruning rules to eliminate unnecessary branches in the search space. The time complexity of their algorithm is $\mathcal{O}(2^n)$. Zhou et al. [24] introduced a novel branching heuristic that improves the running time to $\mathcal{O}(\alpha_k^n)$, where α_k is a constant depending on k and is strictly less than 2. In a related approach, Conte et al. [1] proposed a decomposition-based Bron–Kerbosch algorithm to enumerate k-plexes with diameter at most 2. Although no explicit time complexity is provided, a brief analysis suggests that the algorithm runs in $\mathcal{O}^*(2^{d\Delta})$, where d denotes the degeneracy and Δ the maximum degree of the graph.

Other approaches have also been proposed. In Wang et al. [19], the authors present an algorithm to enumerate all k-plexes with a time complexity of $\mathcal{O}(n^{2k} + n(d\Delta)^{k+1}\alpha_k^d)$. In Dai et al. [11], they describe an algorithm to enumerate all maximal k-plexes of size at least q, with a time complexity of $\mathcal{O}(nr_1^k r_2 \alpha_k^d)$, where $r_1 = \min\left\{\frac{d\Delta}{q^{-2k+2}}, n\right\}$ and $r_2 = \min\left\{\frac{d\Delta^2}{q^{-2k+2}}, n\right\}$. It is also possible to apply the more general framework of Cohen et al. [7], which provides algorithms to enumerate maximal induced subgraphs satisfying hereditary properties. In fact, in a subsequent paper of Berlowitz et al. [2], they applied this framework to the specific case

of k-plexes, resulting in a fixed-parameter tractable polynomial-time delay algorithm with time complexity $\mathcal{O}(f(k)p(n))$ with $p(n)$ a polynomial function of n and $f(k)$ an arbitrary function of k.

Closely related to our problems, the computation of the maximum k-defective clique has also been studied. It is known to be NP-hard [21], and several non-trivial algorithms have been developed to address it [4, 6, 5, 13]. These methods achieve exponential time complexities with base constants strictly less than 2, often derived from the roots of characteristic polynomials. Some offer tighter theoretical bounds, while others emphasize practical performance through preprocessing and reduction techniques. Certain approaches also exploit structural properties of the graph, such as degeneracy, to improve practical runtime. However, even optimized methods become impractical for large values of k, particularly on real-world datasets [13].

3 Our contributions

In this paper, we investigate the enumeration of dense subgraphs under the models of k-plexes and k-defective cliques. More specifically, we focus on connected k-plexes and k-defective cliques of diameter at most 2, as they better capture the notion of a community. To achieve this, Lemma 4 ensures that it suffices to enumerate k-plexes of size at least $2k-1$ and k-defective cliques of size at least $k+2$.

Our main goal is to develop algorithms to enumerate all maximal k-plexes of size at least $2k-1$. We then leverage this approach to enumerate maximal k-defective cliques by exploiting the structural relationship between the two concepts.

We begin by presenting general combinatorial results in Section 4.2. Then, in Section 4.3, we focus on the enumeration of maximal k-plexes. As a starting point, we build on the decomposition introduced in [9]. We first prove that every maximal k-plex of size at least $2k-1$ is contained within a small subgraph of size $\mathcal{O}(d\Delta)$. Using this structural insight, we design an enumeration algorithm that explores each subgraph G_i in time $\mathcal{O}((dk)^3 2^d \Delta^k)$, ensuring that all maximal k-plexes of size at least $2k-1$ are reported without duplication. A subsequent filtering step based on suffix trees guarantees a total enumeration time bounded by $\mathcal{O}(n(dk)^3 2^d \Delta^k)$. This result is formalized in Theorem 16. We also establish an upper bound on the number of such k-plexes, namely $\mathcal{O}(nk2^d\Delta^{k-1})$, and demonstrate the near-optimality of this bound by constructing a family of graphs in which the number of maximal k-plexes of size at least $2k-1$ is $\Omega(n3^{d/3}\Delta^{k-1})$. It follows that our algorithm achieves near-optimal runtime in the worst case.

In Berlow et al. [2], the authors introduced an algorithm for the enumeration of maximal k-plexes of size at least $2k-1$, with fixed-parameter tractable polynomial delay $\mathcal{O}(f(k)p(n))$, where $f(k)$ is an arbitrary function of k, and $p(n)$ is a polynomial in the number of vertices n. The considered parameter is k. We improve upon this result by applying the enumeration to the subgraphs G_i, leveraging the fact that each G_i contains at most $\mathcal{O}(d\Delta)$ vertices. Combined with our upper bound on the total number of solutions across all subgraphs G_i in Lemma 12, this enables the design of a polynomial total algorithm with runtime $\mathcal{O}(\alpha f(k)p(d\Delta))$, where α is the total number of output solutions. To the best of our knowledge, this is the first algorithm with total runtime polynomial both in the output size and in the degeneracy and maximum degree of the input graph.

By applying the same technique to the polynomial delay algorithm of Dai et al. [10] for the enumeration of maximal k-defective cliques of size at least $k+2$, mentioned earlier, we also derive a second output sensitive algorithm with polynomial total runtime. These results are presented in Theorems 17 and 19, respectively.

Finally, we propose an algorithm that lists all maximal 2-plexes in time $\mathcal{O}(n)$ for graphs of bounded degeneracy. This result is presented in Theorem 23. An overview of the existing results, as well as our claimed complexities, can be found in Table 1.

Table 1 Existing algorithms for the problems considered in this paper. Here d is the degeneracy, Δ the maximum degree, n the graph order, N_m the size of the largest k-defective clique, α_k, α_s are constants < 2, and α the output size.

Algorithm	**Output**	**Complexity**
Wu and Pei [20]	Maximal k-plexes	$O(2^n)$
Zhou et al. [24]	Maximal k-plexes	$O(\alpha_k^n)$
Conte et al. [1]	Maximal k-plexes with diameter ≤ 2	$O^*(2^{d\Delta})$
Wang et al. [19]	Maximal k-plexes	$O(n^{2k} + n(d\Delta)^{k+1}\alpha_k^d)$
Dai et al. [11]	Maximal k-plexes of size $\geq q$	$O(nr_1^k r_2 \alpha_k^d)$, where $r_1 = \min\left\{\frac{d\Delta}{q-2k+2}, n\right\}$, $r_2 = \min\left\{\frac{d\Delta^2}{q-2k+2}, n\right\}$
Dai et al. [10]	Maximal k-defective cliques	$O(n^2 N_m^{2k+1}/4^k)$ (*polynomial-time delay*)
Dai et al. [10] (Pivoting)	Maximal k-defective cliques	$O(n\alpha_s^n)$
Dai et al. [10] (Degeneracy-optimized)	Maximal k-defective cliques	$O(n^{k+2}\alpha_k^d)$
Berlowitz et al. [2]	Maximal k-plexes	$O(f(k)\,p(n))$, where $p(n)$ is polynomial in n and $f(k)$ is any function of k (*fixed-parameter tractable polynomial delay*)
This paper	Maximal k-plexes of size $\geq 2k-1$	$O(n(dk)^3 2^d \Delta^k)$
This paper	Maximal k-defective cliques of size $\geq k+2$	$O(\alpha(d\Delta)^2 \frac{(d+k)^{2k+2}}{4^k})$ (*polynomial total time*)
This paper	Maximal 2-plexes	$O(n)$ (when $d = O(1)$)
This paper	Maximal k-plexes of size $\geq 2k-1$	$O(\alpha f(k) p(d\Delta))$, where $p(d\Delta)$ is polynomial in $d\Delta$ and $f(k)$ is any function of k (*fixed-parameter tractable polynomial total time*)

4 Results

4.1 Notations

We consider graphs of the form $G = (V, E)$, which are simple, undirected, with n vertices and m edges. If $X \subseteq V$, the subgraph of G induced by X is denoted by $G[X]$. If $X \subset V$, then X is a proper subgraph of G. When not clear from the context, the vertex set of G will be denoted by $V(G)$. The set $N(x)$ is called the open neighborhood of the vertex x and consists of the vertices adjacent to x in G.

Given an arbitrary ordering $\sigma = v_1, \ldots, v_n$ of the vertices of G, the set V_i consists of the vertices following v_i in this ordering, including v_i itself, that is, $\{v_i, v_{i+1}, \ldots, v_n\}$. In the ordering σ, the rank of v_i, denoted by $\sigma(v_i)$, is its position in the ordering (here, i). By $[n]$, we denote the set of integers $\{1, 2, \ldots, n\}$. The distance between two vertices u and v is the length of the shortest path from u to v. Let $N_i^k(v) = V_i \cap N^k(v)$, where $N^k(v)$ is the set of vertices at distance k from v (in this paper, we consider $N_i^1(v_i) = N(v_i)$).

4.2 Preliminaries

In this section, we present a set of preliminary definitions and combinatorial results that constitute the theoretical foundation of our algorithms. We begin by introducing a decomposition framework based on subgraphs G_i, which plays a central role throughout the paper. We then formally define the two central notions of subgraph density considered in this work: k-plexes and k-defective cliques. These definitions enable a rigorous formulation of the enumeration problems that we aim to solve.

The section proceeds with a series of lemmas that establish the essential properties of these structures. In particular, Lemma 5 and Lemma 6 guarantee that any maximal k-plex of size at least $2k-1$ in a graph G is entirely contained in a unique subgraph G_i of the decomposition, Lemma 11 provides a procedure to verify whether a k-plex is maximal in its corresponding subgraph G_i in time $\mathcal{O}((d+k)\Delta)$, making the approach efficient in practice. These results are central in justifying the correctness and non-redundancy of our enumeration strategy. Further, Lemmas 9 and 10 provide tight bounds on the size and number of such k-plexes in terms of the graph's degeneracy d and maximum degree Δ. In addition, Lemma 12 establishes an upper bound on the total number of such k-plexes in all subgraphs G_i, showing that their sum is proportional to the number of maximal k-plexes in G. This result is crucial in the design and analysis of our output-sensitive algorithms.

▶ **Definition 1.** *Let $G = (V, E)$ be a graph, and let $v_1, \ldots, v_n$ be an ordering of its vertices. For each $i \in [n]$, we define the graph G_i as the subgraph of G induced by the vertex set $V(G_i) = \{v_i\} \cup N_i(v_i) \cup N_i^2(v_i)$, that is, the vertex v_i itself, its neighbors in V_i, and the vertices at distance two from v_i within V_i.*

▶ **Definition 2.** *Let $G = (V, E)$ be an undirected graph and let k be a non-negative integer. A subset $S \subseteq V$ is called a k-plex if, in the induced subgraph $G[S]$, each vertex has at most k non-neighbors (equivalently, at least $|S| - k$ neighbors) within S.*

A k-plex S is said to be maximal in G if there does not exist a strict superset $S' \supset S$ such that S' is also a k-plex in G; that is, S is inclusion-maximal among the k-plexes of G.

▶ **Definition 3.** *Let $G = (V, E)$ be an undirected graph and let k be a non-negative integer. A subset $C \subseteq V$ is called a k-defective clique if the induced subgraph $G[C]$ is missing at most k edges to be a complete clique. A k-defective clique C is said to be maximal in G if there does not exist a strict superset $C' \supset C$ such that C' is also a k-defective clique in G; that is, C is inclusion-maximal among the k-defective cliques of G.*

▶ **Lemma 4** (Two-hop property [18])**.** *If S is a k-plex in G with $|S| \geq 2k-1$, then the subgraph $G[S]$ is connected and has diameter at most two.*

▶ **Lemma 5.** *Let G be a graph, and let $\sigma = (v_1, \ldots, v_n)$ be a degeneracy ordering of its vertices. Let $C \subseteq V(G)$ be a maximal k-plex such that the size of C is at least $2k-1$. Then, there exists a unique index $i \in [n]$ such that $C \subseteq V(G_i)$ and $v_i \in C$.*

Proof. Let C be a maximal k-plex in G with $|C| \geq 2k-1$, and let $i \in [n]$ be the smallest index such that $v_i \in C$. By Lemma 4, the subgraph $G[C]$ has diameter at most two and contains v_i, which implies that every vertex $u \in C \setminus \{v_i\}$ is at distance at most two from v_i. Therefore, each such vertex u is either a neighbor of v_i or is at distance two from v_i, so by the definition of G_i, it follows that $C \subseteq V(G_i)$.

To show uniqueness, suppose for contradiction that there exists an index $j \neq i$ such that $C \subseteq V(G_j)$ and $v_j \in C$. If $j < i$, then $v_j \in C$, which contradicts the minimality of i. If $j > i$, then since $C \subseteq V(G_j)$, we must have $v_i \in V(G_j)$. By definition, $V(G_j) =$

$\{v_j\} \cup N(v_j) \cup N_2(v_j)$, so $v_i \in N(v_j) \cup N_2(v_j)$. However, this is impossible, since all vertices in $N(v_j) \cup N_2(v_j)$ have indices strictly greater than j, whereas $i < j$. Therefore, such an index j cannot exist, proving the uniqueness. ◀

▶ **Lemma 6.** *Let G be a graph, and let $\sigma = (v_1, \ldots, v_n)$ be a degeneracy ordering of its vertices. Let $C \subseteq V(G)$ be a maximal k-plex in subgraph G_i, for some $i \in [n]$, such that the size of C is at least $2k-1$. Then C is not maximal in G if and only if there exists a maximal k-plex $C' \subseteq V(G)$ of size at least $2k-1$, maximal in G, such that $C \subsetneq C' \subseteq V(G_j)$ for some $j < i$.*

Proof. Suppose that C is a maximal k-plex of size at least $2k-1$ in G_i, for some $i \in [n]$, but that C is not maximal in G. Then there exists a set $S \subseteq V(G) \setminus C$ such that the subgraph $G[C \cup S]$ is a k-plex of size at least $2k-1$, and is maximal in G. Let $v_j \in S$ be the vertex with the smallest rank according to the ordering σ.

If $j > i$, then since $G[C \cup S]$ has a diameter of at most 2, the vertex v_j must be at distance at most 2 from every vertex in C. By the definition of G_i, this means $v_j \in V(G_i)$, so $C \cup \{v_j\} \subseteq V(G_i)$ forms a k-plex of size at least $2k-1$ in G_i, contradicting the maximality of C in G_i. Therefore, it must be that $j < i$. Now set $C' = C \cup S$, which by hypothesis is a k-plex of size at least $2k-1$, maximal in G. Since $v_j \in C' \setminus C$, we have $C \subsetneq C'$, and $C' \subseteq V(G_j)$ with $j < i$. This completes the first implication.

Conversely, suppose that there exists a maximal k-plex C' of size at least $2k-1$ in G, included in a subgraph G_j for some $j < i$, such that $C \subsetneq C'$. Then, by definition, C cannot be maximal in G, which completes the proof. ◀

▶ **Corollary 7.** *Let G be a d-degenerate graph and let K be a maximal k-plex of size at least $2k-1$ of an induced subgraph G_i, $i \in [n]$, such that K is not maximal in G. Let C be a maximal k-plex of size at least $2k-1$ of G, which is a subgraph of some graph G_j, $j < i$, and such that K is a subgraph of C. Let $W(K)$ and $W(C)$ be the words obtained from the vertices of the k-plexes K and C ordered according to σ_G. Then $W(K)$ is a proper suffix of $W(C)$.*

Proof. First observe that, by Lemma 6, the k-plex C is well defined and $K \subsetneq C$. Let v_i be the vertex of K with the smallest index in σ_G; thus, v_i appears first in $W(K)$. Since $K \subsetneq C$, we also have $v_i \in C$.

Assume for contradiction that $W(K)$ is not a proper suffix of $W(C)$. This means that there exists at least one vertex $x \in C \setminus K$ which appears after v_i in $W(C)$. Otherwise, $W(K)$ would indeed be a proper suffix of $W(C)$. Let v_j denote the vertex of C with the smallest index in σ_G; according to Lemma 6, we have $j < i$. Since any subset of a k-plex is itself a k-plex, it follows that $K \cup \{x\}$ is a k-plex in G_j. Moreover, as K has size at least $2k-1$, so does $K \cup \{x\}$. By Lemma 4, $K \cup \{x\}$ must have diameter at most 2, implying $x \in N(v_i) \cup N_i^2(v_i)$. Thus, $K \cup \{x\}$ is a k-plex of size at least $2k-1$ in G_i, which contradicts the maximality of K in G_i. ◀

▶ **Lemma 8.** *Let G be a graph of degeneracy d. Then the size of any maximal k-plex in G of size at least $2k-1$ is at most $d+k$.*

Proof. Let C be a maximal k-plex in G of size at least $2k-1$, and let σ be a degeneracy ordering of G with degeneracy d. Let v_i be the vertex of C with the smallest rank in σ_G. By Lemma 5, $C \subseteq V(G_i)$. Therefore, C can contain at most $k-1$ vertices in $N_i^2(v_i)$; otherwise, the number of v_i's non-neighbors would exceed the allowed $k-1$, contradicting the definition of a k-plex. Moreover, by the definition of degeneracy, v_i has at most d neighbors in V_i, so at most d vertices in $N_i(v_i)$ can belong to C. Since $N_i(v_i)$ and $N_i^2(v_i)$ are disjoint, we obtain the following inequality: $|C| \leq d + k$. ◀

▶ **Lemma 9.** *Let G be a graph, and d its degeneracy. Then the number of maximal k-defective cliques of size at least $k+2$ is $\mathcal{O}(nk2^d\Delta^{k-1})$.*

Proof. Let G be a d-degenerate graph and σ a degeneracy ordering of its vertices. Denote by $\mathcal{P}([d])$ the set of all subsets of $[1,d]$, and by $\mathcal{P}_{k-1}$ the set of all subsets of $N_i^2(v_i)$ of size at most $k-1$. Let α_i denote the number of maximal k-plex of size at least $2k-1$ in the graph G_i, for $i \in [n]$, containing v_i, and let α be the number of maximal k-plex of size at least $2k-1$ in the graph G. According to Lemma 5, for every maximal k-plex of size at least $2k-1$ in G, there exists $i \in [n]$ such that $C \subseteq V(G_i)$ and $v_i \in C$. Furthermore, by Lemma 6, there may exist a maximal k-plex of size at least $2k-1$ in G_i containing v_i (for some $i \in [n]$), which are not maximal in G. It follows that $\alpha \leq \sum_{i=1}^{n} \alpha_i$.

Let $i \in [n]$. By definition of σ, the vertex v_i has at most d neighbors of higher rank (i.e., greater index) in σ; in other words, the cardinality of $N(v_i)$ is at most d. Each vertex in $N(v_i)$ is adjacent to at most Δ vertices in $N_{2i}(v_i)$, so $|N_{2i}(v_i)| \leq d\Delta$. Moreover, by arguments analogous to those in the proof of Lemma 8, any maximal k-plex of size at least $2k-1$ in G_i containing v_i contains at most d vertices from $N(v_i)$ and at most $k-1$ vertices from $N_{2i}(v_i)$. Thus, for k-plex C, there exist two sets $S_1 \subseteq \mathcal{P}([d])$ and $S_2 \subseteq \mathcal{P}_k$ such that $C = \{v_i\} \cup S_1 \cup S_2$. Therefore, we obtain the following inequality: $\alpha_i \leq |\mathcal{P}([d]) \times \mathcal{P}_{k-1}|$, with $|\mathcal{P}_{k-1}| = \sum_{r=0}^{k-1} \binom{d\Delta}{r}$ and $|\mathcal{P}([d])| = 2^d$. As $\sum_{r=0}^{k-1} \binom{d\Delta}{r} \in \mathcal{O}(k\Delta^{k-1})$, it follows that $|\mathcal{P}([d]) \times \mathcal{P}_{k-1}| \in \mathcal{O}(k2^d\Delta^{k-1})$, and thus $\alpha_i \in \mathcal{O}(k2^d\Delta^{k-1})$. Consequently, $\alpha \leq \sum_{i=1}^{n} \alpha_i$ which is bounded by $\mathcal{O}(nk2^d\Delta^{k-1})$. ◀

▶ **Lemma 10.** *For any tuple (k,d,Δ,n) such that $d \leq k \leq \Delta \leq n$, we can construct a graph of order n with maximum degree Δ and degeneracy d such that G has $\Omega\left((n-d-\Delta)3^{d/3}\Delta^{k-1}\right)$ maximal k-plexes of size at least $2k-1$.*

Proof. Let $G = (V,E)$ be a simple, undirected, d-degenerate graph with maximum degree Δ, and let $\sigma = (v_1, v_2, \ldots, v_n)$ be a degeneracy ordering. Set $S_1 = \{v_i : 1 \leq i \leq n-d-\Delta\}$, $S_2 = V \setminus S_1$, and consider a partition $S_2 = S_2^1 \cup S_2^2$ such that $|S_2^1| = d$, $G[S_2^2]$ is a clique, and S_1 is an independent set.

For every $i \in \{1, \ldots, n-d-\Delta\}$, let G_i be the subgraph induced by the vertex set $\{v_i\} \cup S_2$, where v_i is adjacent to all vertices in S_2^1, and v_i is not adjacent to any vertex in S_2^2. Let α_i denote the number of maximal k-plexes of size at least $2k-1$ in G_i, and let α denote the number of such k-plexes in G. For $i \in \{1, \ldots, n-d-\Delta\}$, Consider $C = \{v_i\} \cup C_1 \cup C_2$, where C_1 is a maximal clique of S_2^1 and $C_2 \subseteq S_2^2$ is a subset of cardinality $k-1$. We show that C is a maximal k-plex of size at least $2k-1$ in G_i. First, $C \subseteq V(G_i)$, and since $v_i \notin C_2$, we have $|C| = d + k \geq k+2$ by Lemma 8. Moreover, since S_2^2 induces a clique and v_i is not adjacent to any vertex in S_2^2, the subgraph $G_i[C]$ missing exactly $k-1$ edges, namely those between v_i and each vertex of C_2. Thus, C is indeed a k-plex in G_i.

Assume for contradiction that C is not maximal in G; then there exists $u \in V \setminus C$ such that $C \cup \{u\}$ is a k-defective clique. Since $|S_2^1| = d$, then v_i and u are not adjacent; hence, v_i has exactly k non-neighbors in $G[C \cup \{u\}]$, contradicting the definition of a k-plex. C is therefore maximal both in G and in G_i.

For every choice of a pair (C_1, C_2), we obtain a distinct maximal k-plex. There are exactly $3^{d/3}$ maximal cliques in S_2^1 and $\binom{|S_2^2|}{k-1} = \binom{\Delta}{k-1}$ subsets of size k in S_2^2 (since $|S_2^2| = \Delta$). Thus, we obtain $3^{d/3}\binom{\Delta}{k-1}$ maximal k-plexes of the form C. Therefore, for each $i \in \{1, \ldots, n-d-\Delta\}$, we have $\alpha_i \geq 3^{d/3}\binom{\Delta}{k-1}$. Moreover, since S_1 is an independent set, every maximal k-plex of size at least $2k-1$ in G_i is also maximal in G, for each such i.

Finally, we obtain: $\alpha = \sum_{i=1}^{n-d-\Delta} \alpha_i \geq (n-d-\Delta)3^{d/3}\binom{\Delta}{k-1}$ Now, since $\binom{\Delta}{k-1} \in \Omega(\Delta^{k-1})$, it follows that $\alpha \in \Omega((n-d-\Delta)3^{d/3}\Delta^{k-1})$. ◀

▶ **Lemma 11.** *Let G be a d-degenerate graph with maximum degree Δ, σ a degeneracy ordering of its vertices, and G_i the subgraph defined as in Definition 1 for some $i \in [n]$. If $C \subseteq V(G_i)$ is a k-plex of size at least $2k-1$, then one can verify whether C is maximal in G_i in time $\mathcal{O}((d+k)d\Delta)$.*

Proof. Let $C \subseteq V(G_i)$ be a k-plex of size at least $2k-1$. To check whether C is maximal in G_i, it suffices, by Definition 2, to verify whether there exists a vertex $u \in V(G_i) \setminus C$ such that $C \cup \{u\}$ is still a k-plex in G_i. The cardinality of $V(G_i)$ is bounded by $1 + d + d\Delta$ (i.e., $O(d\Delta)$), since G_i contains at most d direct neighbors of v_i and $d\Delta$ vertices at distance two. Therefore, the size of $V(G_i) \setminus C$ is also $O(d\Delta)$. For each $u \in V(G_i) \setminus C$, one needs to check whether $|C \cup \{u\}| - \deg_{G[C\cup\{u\}]}(v) \leq k-1$, for each $v \in C \cup \{u\}$, which can be done in $O(d+k)$ time, since $|C| \leq d+k$ by Lemma 4. If this condition holds for every $v \in C \cup \{u\}$), then C is not maximal in G_i. Therefore, the maximality of a k-plex can be verified in $O((d+k)d\Delta)$ time. ◀

▶ **Lemma 12.** *Let G be a d-degenerate graph, and let G_i, $i \in [n]$, be the family of induced subgraphs defined in Definition 1. Let α denote the number of maximal k-plexes of size at least $2k-1$ in G, and let α_i denote the number of such k-plexes in G_i. We have that $\sum_{i=1}^{n} \alpha_i \leq (d+k)\alpha$.*

Proof. Let m_i denote the number of maximal k-plexes of size at least $2k-1$ in G_i, $i \in [n]$ that are also maximal in G, and let Nm_i be the number of such k-plexes in G_i that are not maximal in G. Thus, $\alpha_i = m_i + Nm_i$.

Let C be a maximal k-plex of size at least $2k-1$ in G. By Lemma 8, the number of vertices in C is at most $d+k$. Consequently, $W(C)$, the word formed by the vertices of C ordered according to σ_G, can have at most $d+k-1$ proper suffixes.

Moreover, by Corollary 7, every maximal k-plex K of size at least $2k-1$ in some subgraph G_i, $i \in [n]$, that is not maximal in G, satisfies $W(K)$ being a proper suffix of $W(C)$ for some maximal k-plex C in G. Therefore, $\sum_{i=1}^{n} Nm_i \leq (d+k-1)\alpha$, and it follows that $\sum_{i=1}^{n} \alpha_i = \sum_{i=1}^{n}(m_i + Nm_i) \leq \sum_{i=1}^{n} m_i + \sum_{i=1}^{n} Nm_i \leq \alpha + (d+k-1)\alpha = (d+k)\alpha$. ◀

4.3 Algorithms for the enumeration of k-plexes

We now describe our enumeration framework for computing all maximal k-plexes of size at least $2k-1$ in a graph G. This section presents two algorithms: a base algorithm that enumerates all such k-plexes in each subgraph G_i, and a refined version that guarantees uniqueness of output using a generalized suffix tree. The first algorithm operates locally in each G_i and its enumeration time depends solely on the degeneracy of G, ensuring efficient exploration of the search space. The second algorithm builds upon the first by incorporating a filtering mechanism based on suffix tree rejection to avoid duplicate outputs. Both algorithms are described in detail and formally analyzed. The correctness and non-redundancy of the enumerated solutions are established in Theorems 13 and 14, while their complexities are proved in Theorems 15 and 16. Finally, in Theorem 17, we derive an output-sensitive result by applying the polynomial delay algorithm from [12] to the subgraphs G_i defined in Definition 1. By exploiting the fact that each G_i has size $\mathcal{O}(d\Delta)$, and combining this with the bound established in Lemma 12 on the total number of maximal k-plexes across all G_i's, we design an output-sensitive algorithm with a total running time polynomial in $d\Delta$, and linear in the number of solutions.

Algorithm 1 The procedure to find the local maximal k-plexes.

Data: A graph G_i, integer k
Result: All maximal k-plexes of size at least $2k-1$ in G_i containing v_i
1 Initialize $C \leftarrow \emptyset$;
2 Let $N_1 \leftarrow N(v_i)$;
3 Let $N_2 \leftarrow N_i^2(v_i)$;
4 **for** *every subset* $S_1 \subseteq N_1$ **do**
5 **for** *every subset* $S_2 \subseteq N_2$ *such that* $|S_2| \leq k-1$ **do**
6 Let $C \leftarrow \{v_i\} \cup S_1 \cup S_2$;
7 **if** $|C| \geq 2k-1$ *and* $G_i[C]$ *is a* k*-plex* **then**
8 **if** *no proper superset of* C *in* G_i *satisfies the same properties* **then**
9 Add C to the output;
10 **return** All cliques C found;

Algorithm 2 The algorithm to find all maximal k-plexes.

Data: A graph G, integer k
Result: All maximal k-plexes of size at least $2k-1$ in G
1 Compute d the degeneracy of G and σ_G;
2 Initialize T an empty generalized suffix tree;
3 **for** $i = 1$ *to* n **do**
4 Algorithm 1(G_i, k)
5 **for** *every maximal* k*-plex* C *of graph* G_i **do**
6 Order the vertices of C following σ_G;
7 Search for C in T;
8 **if** *there is a match* **then**
9 Reject it;
10 **else**
11 Insert the proper suffixes of C in T;
12 Output C;

4.3.1 Proof of correctness

▶ **Theorem 13.** *Given a subgraph G_i as defined in Definition 1, Algorithm 1 outputs exactly all maximal k-plexes of size at least $2k-1$, containing v_i, without duplication.*

Proof. Assume for contradiction that there exists a k-plex C of size at least $2k-1$, containing v_i and maximal in G_i, which is not reported by Algorithm 1. Let $C_1 = C \cap N_1$ and $C_2 = C \cap N_2$, so that $C = \{v_i\} \cup C_1 \cup C_2$. Observe that since $C_1 \subseteq N_1$, the subset $S_1 = C_1$ is examined at line 4 of the algorithm. Moreover, as v_i has no neighbors in C_2, it follows that $|C_2| \leq k-1$. Thus, the subset $S_2 = C_2$ is included in the enumeration at line 5. Consequently, the set C is constructed at line 6, and by assumption, both the size and k-plex constraints at line 7, as well as the maximality criterion at line 8, are satisfied. Hence, Algorithm 1 must output C, a contradiction. It follows that every k-plex of size at least $2k-1$, containing v_i and maximal in G_i, is indeed generated by the algorithm.

We now show that Algorithm 1 does not produce any duplicates. Suppose, to the contrary, that a k-plex C is generated more than once, i.e., there exist distinct pairs $(S_1, S_2) \neq (S_1', S_2')$ such that $C = \{v_i\} \cup S_1 \cup S_2 = \{v_i\} \cup S_1' \cup S_2'$. By construction, $S_1, S_1' \subseteq N_1$, $S_2, S_2' \subseteq N_2$, with $N_1 \cap N_2 = \varnothing$. Thus, the decomposition $C = \{v_i\} \cup S_1 \cup S_2$ is unique, as the sets S_1 and S_2 are disjoint and correspond to non-overlapping neighborhoods. Therefore, if $(S_1, S_2) \neq (S_1', S_2')$, the resulting sets are necessarily distinct, contradicting our assumption. This establishes that each clique is output at most once by the algorithm. ◀

▶ **Theorem 14.** *Given a subgraph G, and an integer k, Algorithm 2 outputs exactly all maximal k-plexes of size at least $2k-1$, without duplication.*

Proof. Assume for contradiction, that there exists a maximal k-plex C of size at least $2k-1$ in G that is not output by Algorithm 2. By Lemma 5, there exists a unique index $i \in [n]$ such that C is maximal in G_i and $C \subseteq V(G_i)$. According to Theorem 13, Algorithm 1, which is called at line 4 of Algorithm 2, enumerates all maximal k-plexes in G_i without duplication. Thus, C must necessarily be produced at line 5. It follows that C can only be rejected by the suffix-tree rejection test (lines 7–9) if it is detected as a proper suffix of some already produced k-plex C' that is maximal in G, with $C \subsetneq C'$, as ensured by Corollary 7. However, the inclusion $C \subsetneq C'$ contradicts the maximality of C in G. Therefore, C must indeed be output by Algorithm 2. We conclude that Algorithm 2 outputs all maximal k-plexes of size at least $2k-1$ in G.

We now show that there is no duplication. Assume for contradiction, that this is not the case; that is, there exists a maximal k-plex C of size at least $2k-1$ in G that is output at least twice by Algorithm 2. Since Algorithm 1 enumerates, without duplication, all maximal k-plexes of each subgraph G_i, $i \in [n]$, at line 4, the only possibility is that C is enumerated in two different subgraphs G_i and G_j with $i \neq j$. However, this contradicts Lemma 5, which guarantees that a maximal k-plex of size at least $2k-1$ in G is included in exactly one such subgraph G_i.

Hence, by contradiction, Algorithm 2 outputs exactly all maximal k-plexes of size at least $2k-1$ in G, without duplication. ◀

4.3.2 Proofs of complexity

This subsection establishes the computational complexity of our enumeration algorithms for maximal k-plexes of size at least $2k-1$. We analyze both the local enumeration within individual subgraphs G_i and then the enumeration across the entire graph G. These bounds exploit the structural properties of degeneracy orderings and leverage the combinatorial limits established in Lemma 9, combined with efficient suffix tree operations to ensure uniqueness of the output.

▶ **Theorem 15.** *Given a graph G_i, $i \in [n]$, and an integer k, the time complexity of Algorithm 1 is in $\mathcal{O}((dk)^3 2^d \Delta^k)$.*

Proof. Let G be a d-degenerate graph, and let σ be a degeneracy ordering of its vertices. For each vertex v_i, the algorithm operates on the subgraph G_i, as defined in Definition 1. The `for` loop at line 4 enumerates all subsets $S_1 \subseteq N_1$, yielding 2^d iterations. The loop at line 5 iterates over all $S_2 \subseteq N_2$ with $|S_2| \leq k-1$, corresponding to exactly $\sum_{r=0}^{k-1} \binom{d\Delta}{r}$ iterations, which is $\mathcal{O}(k\Delta^{k-1})$. The checks at line 7 are performed in $\mathcal{O}(d+k)$ time, and the maximality verification at line 8 requires $\mathcal{O}(d(d+k)\Delta)$ time according to Lemma 11. Therefore, the body of the inner loop runs in $\mathcal{O}(d(d+k)^2\Delta$ time. In total, the algorithm has a complexity of $\mathcal{O}(dk(d+k)^2\Delta 2^d \Delta^{k-1}) = \mathcal{O}(dk(d+k)^2 2^d \Delta^k)$, that is, $\mathcal{O}((dk)^3 2^d \Delta^k)$. ◀

▶ **Theorem 16.** *Given a graph G with degeneracy d and maximum degree Δ, and an integer k, the time complexity of Algorithm 2 is in $\mathcal{O}(n(dk)^3 2^d \Delta^k)$.*

Proof. Let G be a d-degenerate graph and let σ be a degeneracy ordering of its vertices. The computation of the degeneracy and the ordering σ at line 1 can be performed in $\mathcal{O}(m)$ time, where m is the number of edges. For lines 3–4, we consider each subgraph G_i for $i \in [n]$, so this step is executed for n subgraphs in total. Enumerating all maximal k-plexes of size at least $2k-1$ in each G_i takes $\mathcal{O}((dk)^3 2^d \Delta^k)$ time by Theorem 15. According to Lemma 9, each subgraph G_i contains $\mathcal{O}(2^d \Delta^k)$ maximal k-plexes of size at least $2k-1$. Thus, the `for` loop at line 5 runs $\mathcal{O}(2^d \Delta^k)$ times. Regarding lines 5–12: Ordering the vertices of C according to σ can be done in $\mathcal{O}(|C|) \subseteq \mathcal{O}(d+k)$ time, by Lemma 8. Suffix tree operations (search and insertion) for words of length at most $d+k$ can be performed in $\mathcal{O}(d+k)$ time (see [17]). Thus, the body of the inner loop at line 5 has complexity $\mathcal{O}((d+k)2^d \Delta^k)$, and so the outer loop at line 3 has total complexity $\mathcal{O}\big(n((dk)^3 2^d \Delta^k + n(d+k)2^d \Delta^k)\big) = \mathcal{O}(n(dk)^3 2^d \Delta^k)$. Since all other steps require at most $\mathcal{O}(m)$ operations, it follows that the overall complexity of the algorithm is $\mathcal{O}(n(dk)^3 2^d \Delta^k)$. ◀

▶ **Theorem 17.** *Given a graph G of degeneracy d and maximum degree Δ, and an integer k, one can enumerate all maximal k-plexes of size at least $2k-1$, without duplication, in time $\mathcal{O}\left(\alpha\, f(k)\, p(d\Delta)\right)$, where α denotes the number of maximal k-plexes of size at least $2k-1$ in G, $p(d\Delta)$ is a polynomial function of $d\Delta$, and $f(k)$ is an arbitrary function of k. The algorithm is output-sensitive, with total running time proportional to the number of solutions.*

Proof. Let α denote the number of maximal k-plexes of size at least $2k-1$ in G, and let α_i denote the number of such k-plexes in the subgraph G_i, for $i \in [n]$. In [2], the authors applied this framework to the specific case of k-plexes, resulting in a fixed-parameter tractable algorithm with polynomial delay and time complexity $\mathcal{O}(f(k)\, p(n))$, where n is the order of the graph, $p(n)$ is a polynomial function of n, and $f(k)$ is an arbitrary function of k. This algorithm can be adapted to our decomposition into subgraphs G_i, which yields a significant improvement in complexity: $\mathcal{O}\big(\alpha\, f(k)\, p(d\Delta)\big)$. More concretely, the approach consists of applying this algorithm to each subgraph G_i, for $i \in [n]$. For each $i \in [n]$, the order of G_i is $\Theta(d\Delta)$, where Δ is the maximum degree of G. Consequently, the enumeration in each G_i can be performed with a delay between two outputs in $\mathcal{O}\big(f(k)\, p(d\Delta)\big)$. Since there are α_i such cliques in G_i, for $i \in [n]$, the enumeration in G_i takes $\mathcal{O}\left(\alpha_i f(k)\, p(d\Delta)\right.$ time.

Moreover, by applying the same arguments used in the proof of Lemma 12, it can be shown that $\sum_{i=1}^{n} \alpha_i \leq (d+k-1)\alpha$. It follows that the enumeration over all subgraphs G_i can be performed in total time $\mathcal{O}\left(f(k)\, p(d\Delta) \sum_{i=1}^{n} \alpha_i\right) \in \mathcal{O}\left(\alpha f(k)\, p(d\Delta)\right)$. To ensure that only cliques which are maximal in G are retained, a similar filtering procedure to that of Algorithm 2 can be used. This step requires $\mathcal{O}\big((d+k)\alpha\big)$ time, as established in the proof of Theorem 16. Therefore, by Lemma 5, Lemma 6, and Corollary 7, this approach yields an exhaustive, duplication-free enumeration of all maximal k-defective cliques of size at least $k+2$ in G, with total time complexity $\mathcal{O}\left(\alpha f(k)\, p(d\Delta)\right)$. ◀

4.4 Application to the enumeration of k-defective cliques

In this section, we show how to effectively extend our analytical framework to the enumeration of k-defective cliques. We leverage the fact that a k-defective clique is, by definition, a $(k+1)$-plex, which allows us to reuse the algorithms previously developed for the enumeration of k-plexes. By inserting an additional verification step into the enumeration algorithm, we

ensure that only those $(k+1)$-plexes satisfying the k-defectiveness condition are retained. This refinement enables the duplication-free generation of all maximal k-defective cliques of size at least $k+2$ in a given graph, as established in Theorem 16.

We then introduce an alternative approach based on an output-sensitive algorithm that fully exploits the degeneracy and the maximum degree of the graph to improve the computational complexity. Theorem 19 formalizes this optimization by providing a bound on the total running time in terms of the number of solutions. To the best of our knowledge, these are the first output-sensitive bounds for this problem that are expressed solely in terms of the parameters k, Δ, d, and the number of maximal k-defective cliques. Since the proofs of Theorems 18 and 19 are similar to the proofs presented in Section 4.3.

▶ **Theorem 18.** *Given a graph G of degeneracy d and maximum degree Δ, and an integer k, one can enumerate all maximal k-defective cliques of size at least $k+2$, without duplication, in time $\mathcal{O}\left(n(dk)^5 2^d \Delta^{k+1}\right)$.*

Proof. Since every k-defective clique is a $(k+1)$-plex, the enumeration algorithm for maximal $(k+1)$-plexes can be leveraged to enumerate all maximal k-defective cliques. Specifically, Algorithm 2 is applied to enumerate all maximal $(k+1)$-plexes of size at least $2k+1$. Immediately after Line 11 of Algorithm 2, an additional verification step is incorporated to test whether the current $(k+1)$-plex is also a k-defective clique. This verification consists in counting the number of missing edges in the subgraph induced by the current $(k+1)$-plex. It can be performed in $\mathcal{O}((d+k)^2)$ time. According to Theorem 16, Algorithm 2 enumerates all maximal $(k+1)$-plexes of size at least $2k+1$ in $\mathcal{O}\left(n(dk)^3 2^d \Delta^{k+1}\right)$ time. Since the additional verification for the k-defective clique condition is performed once per candidate and requires at most $\mathcal{O}((d+k)^2)$ time, the total time per candidate becomes $\mathcal{O}\left((dk)^3(d+k)^2 2^d \Delta^{k+1}\right) = \mathcal{O}\left((dk)^5 2^d \Delta^{k+1}\right)$. Finally, by Lemma 9, Lemma 10, and Corollary 7, the suffix-tree-based filtering guarantees that only the k-defective cliques which are maximal in G are retained, and that no duplicates are produced. Hence, this approach yields an exhaustive, duplication-free enumeration of all maximal k-defective cliques of size at least $k+2$ in G, with overall time complexity $\mathcal{O}\left((dk)^5 2^d \Delta^{k+1}\right)$. ◀

▶ **Theorem 19.** *Given a graph G of degeneracy d and maximum degree Δ, and an integer k, one can enumerate all maximal k-defective cliques of size at least $k+2$, without duplication, in time $\mathcal{O}\left(n\alpha(d\Delta)^2 \frac{(d+k)^{2k+2}}{4^k}\right)$, where α denotes the number of maximal k-defective cliques of size at least $k+2$ in G. The algorithm is output-sensitive, with total running time proportional to the number of solutions.*

Proof. Let α denote the number of maximal k-defective cliques of size at least $K+2$ of G, and let α_i denote the number of maximal k-defective cliques of size at least $k+2$ in the subgraph G_i, $i \in [n]$. In [10], Algorithm 3 enumerates all maximal k-defective cliques of size at least $k+2$ in a graph G, with a delay of $\mathcal{O}\big(n^2 N_m^{2k+1}/4^k\big)$ between two outputs, where n is the order of the graph and N_m denotes the size of the largest maximal k-defective clique in G. This algorithm can be adapted to our decomposition into subgraphs G_i, which allows for a significant complexity improvement to $\mathcal{O}\big(n(d\Delta)^2(d+k)^{2k+2}/4^k\big)$, where d is the degeneracy of the input graph and n its order. More concretely, the approach consists in applying Algorithm 3 to each subgraph G_i, $i \in [n]$. For every $i \in [n]$, the order of G_i is $\Theta(d\Delta)$, where Δ is the maximum degree of G. Moreover, Lemma 4 ensures that the size of the largest maximal k-defective clique of size at least $k+2$ in G_i is at most $d+k+1$. Thus, the enumeration in each G_i can be performed with a polynomial delay of $\mathcal{O}\big((d\Delta)^2(d+k)^{2k+1}/4^k\big)$. Consequently, the enumeration in each G_i can be performed

with a polynomial delay between two outputs in $\mathcal{O}\left((d\Delta)^2 \frac{(d+k)^{2k+1}}{4^k}\right)$. Since there are α_i such cliques in G_i, for $i \in [n]$, the enumeration in G_i takes $\mathcal{O}\left(\alpha_i (d\Delta)^2 \frac{(d+k)^{2k+1}}{4^k}\right)$ time. Moreover, by applying exactly the same arguments used in the proof of Lemma 12, it can be shown that $\sum_{i=1}^{n} \alpha_i \leq (d+k-1)\alpha$. It follows that the enumeration over all subgraphs G_i can be performed in total time $\mathcal{O}\left((d\Delta)^2 \frac{(d+k)^{2k+1}}{4^k} \sum_{i=1}^{n} \alpha_i\right) \in \mathcal{O}\left(\alpha (d\Delta)^2 \frac{(d+k)^{2k+2}}{4^k}\right)$. To ensure that only cliques which are maximal in G are retained, one can use a similar filtering procedure as in Algorithm 2, which requires $\mathcal{O}\big(\alpha(d+k)\big)$ time, as established in the proof of Theorem 16. Therefore, by Lemma 5, Lemma 6, and Corollary 7, this approach yields an exhaustive, duplication-free enumeration of all maximal k-defective cliques of size at least $k+2$ in G with$\mathcal{O}\big(\alpha (d\Delta)^2 \frac{(d+k)^{2k+3}}{4^k}\big)$ time. ◀

4.5 Application to the enumeration of 2-plexes

In this section, we focus on the enumeration of maximal 2-plexes. We build upon some of the combinatorial results presented in the previous sections and apply a variation of a result by Eppstein [12] concerning the enumeration of non-induced maximal bicliques. This allows us to derive a linear-time algorithm, with complexity $\mathcal{O}(n)$, to enumerate maximal 2-plexes in graphs of constant degeneracy.

Recall that a non-induced biclique is a bipartite subgraph $A \times B$ such that every vertex in A is adjacent to every vertex in B, without the requirement that A and B form independent sets. Such a biclique is said to be maximal if it is not properly contained in any other biclique of the same form. Moreover, given an undirected graph G, a d-bounded orientation of G is any orientation of the edges such that each vertex has out-degree at most d. In this section, the term biclique always refer to non-induced bicliques.

▶ **Lemma 20.** *Let G be an undirected d-degenerate graph, and let C be a 2-plex of size at least 3 and maximal in G. Then there exists a maximal non-induced biclique $A \times B$ in G such that $C \subseteq A \cup B$.*

Proof. Let G be an undirected d-degenerate graph, and let C be a maximal 2-plex of size at least 3 in G. By Lemma 5, there exists $i \in [n]$ such that $C \subseteq V(G_i)$ and $v_i \in C$. Let us define $A = C \cap N(v_i)$, and $B = C \cap (N_i^2(v_i) \cup \{v_i\})$, so that $C = A \cup B$. By construction, $A \subseteq N(v_i)$, and thus $|A| \leq d$. Since C is a 2-plex, C can contain at most one vertex in $N_i^2(v_i)$, ensuring that v_i has at most one non-neighbor in C. Hence, $|B| \leq 2$. Furthermore, by the definition of $N(v_i)$, the vertex v_i is adjacent to every vertex in A, and every $u \in B$ is adjacent to every vertex in A as well; otherwise, u would have more than one non-neighbor in C, contradicting the fact that C is a 2-plex. Consequently, $A \times B$ forms a biclique. If $A \times B$ is maximal, the lemma holds, otherwise it is included in a maximal non-induced biclique containing $A \times B$. ◀

▶ **Lemma 21** (Folklore)**.** *If a graph G has degeneracy d, it has a d-bounded orientation.*

▶ **Lemma 22** (Eppstein [12])**.** *Given a graph G with a d-bounded orientation, and a collection of sets A_i, where each A_i is a 2-plex of size at most d in G, it is possible to compute the corresponding non-induced bicliques $A_i \times B_i$ in total time $\mathcal{O}(d2^d n + d^2 r)$, where r denotes the total size of all sets A_i.*

Proof. We rely on a result established in [12], which shows that, given a graph equipped with a d-bounded orientation and a collection of sets $A_1, \ldots, A_m$, where each A_i is a subset of vertices, all non-induced bicliques $A_i \times B_i$ can be listed in total time $\mathcal{O}(d^2 n + d^2 r)$,

where r denotes the total size of sets A_i. In our case, we consider as candidates the sets A_i defined as 2-plexes of size at most d in the subgraphs G_j, $j \in [n]$, according to Definition 1. More specifically, we restrict to sets $A_i \subseteq N(v_j)$, since $N(v_j)$ contains at most d vertices due to the degeneracy of the graph. Therefore, each G_j contains at most 2^d such subsets. Summing over all n vertices v_j, the total number of sets A_i is thus bounded by $n2^d$. All these sets can therefore be listed in time $\mathcal{O}(n2^d)$. Then, by applying the result from [12] to this collection, all non-induced bicliques $A_i \times B_i$ can be listed in total time $\mathcal{O}(d^2n + d^2n2^d + n2^d) = \mathcal{O}(d^2n + d^2 2^d n)$, since $r = n2^d$. ◀

The proof of the following theorem is similar to that given by Eppstein [12] for their linear-time algorithm to enumerate maximal non-induced bicliques, with a few additional simple steps.

▶ **Theorem 23.** *If an undirected graph G has degeneracy $d = \mathcal{O}(1)$, then all maximal 2-plexes of G can be listed in $\mathcal{O}(n)$ time.*

Proof. We rely on Theorem 1 from [12], which establishes that for any graph G with arboricity $a = \mathcal{O}(1)$, all maximal non-induced bicliques of G can be listed without duplication in total time $\mathcal{O}(n)$, where n denotes the order of the graph. In their proof, the arboricity is used to ensure the existence of a $2a$-bounded acyclic orientation of the graph. However, in our context, Lemma 21 guarantees that any d-degenerate graph admits a d-bounded orientation. Moreover, Lemma 3 from [12] ensures that such a d-bounded orientation implies the existence of a $2d$-bounded acyclic orientation, which can be computed in linear time. It follows that a graph with degeneracy $d = \mathcal{O}(1)$ admits a $2d$-bounded acyclic orientation, thus allowing the application of Theorem 1 from [12] within our framework. To make use of this result, we adapt the technical lemmas from [12] by replacing their Lemmas 2 and 4 with our Lemmas 21 and 22, respectively. Indeed, in the proof of Lemma 21, we define a collection of sets A_i as the 2-plexes of size at most d in the subgraphs G_j, which enables the efficient listing of non-induced bicliques of the form $A_i \times B_i$. Then, Lemma 20 ensures that every maximal 2-plex in G is included in some maximal biclique of G.

However, this is not sufficient to recover exactly all maximal 2-plexes. Recall that in Lemma 20, it is shown that a maximal 2-plex is composed of a vertex v_i (the vertex with the smallest rank in the 2-plex, with respect to a degeneracy ordering), a set A_i of its neighbors with higher rank at distance one, and a single vertex with the highest rank at distance two. Therefore, given a maximal non-induced biclique $A_i \times B_i$, to find all 2-plexes included in that biclique, it suffices to identify the vertices in B_i that are not neighbors of the vertex v_i. This can be done in time $\mathcal{O}(n \times d)$. According to Theorem 1 of Eppstein [12], the computation of the sets B_i requires $\mathcal{O}(a^3 2^{2a} n)$ time, where a is the arboricity of the graph. Recalling that $d = 2a$, the total time complexity becomes $\mathcal{O}(a^3 2^{2a} n + d2^{2a} n) = \mathcal{O}((a^3 + 2a)2^{2a} n)$, which yields the claimed result. ◀

References

1 A.Conte, T. De Matteis, D. De Sensi, R. Grossi, A. Marino, and L. Versari. D2k: Scalable community detection in massive networks via small-diameter k-plexes. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1272–1281. ACM, 2018. doi:10.1145/3219819.3220093.

2 D. Berlowitz, S. Cohen, and B. Kimelfeld. Efficient enumeration of maximal k-plexes. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 431–444. ACM, 2015. doi:10.1145/2723372.2746484.

3 C. Bron and J. Kerbosch. Algorithm 457: Finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 1973. doi:10.1145/362342.362367.

4 Y. Cai and M. Xiao. A new exact algorithm for the maximum k-defective clique problem, 2023. arXiv:2309.02635.

5 Y. Cai and M. Xiao. A simple yet effective branching algorithm for the maximum k-defective clique problem, 2024. arXiv:2407.16588.

6 X. Chen, Y. Zhou, J.-K. Hao, and M. Xiao. Computing maximum k-defective cliques in massive graphs. *Computers & Operations Research*, 127:105131, 2021. doi:10.1016/J.COR.2020.105131.

7 S. Cohen, B. Kimelfeld, and Y. 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.2007.12.006.

8 A. Conte, P. Foggia, and M. Vento. Exploring community structures using relaxed cliques. *European Journal of Operational Research*, 226(1):103–113, 2013. doi:10.1016/j.ejor.2012.10.031.

9 A. Conte, T. De Matteis, D. De Sensi, R. Grossi, A. Marino, and L. Versari. D2k: Scalable community detection in massive networks via small-diameter k-plexes. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1272–1281. ACM, 2018. doi:10.1145/3219819.3220101.

10 Q. Dai, R.-H. Li, M. Liao, and G. Wang. Maximal defective clique enumeration. *Proceedings of the ACM on Management of Data*, 1(1):Article 77, 2023. doi:10.1145/3588931.

11 Q. Dai, R.-H. Li, and G. Wang. Output-sensitive enumeration of maximal k-plexes of minimum size in massive graphs, 2024. arXiv:2402.13008.

12 D. Eppstein. Arboricity and bipartite subgraph listing algorithms. *Information Processing Letters*, 51(4):207–211, 1994. doi:10.1016/0020-0190(94)90121-X.

13 T. Gao, Z. Xu, R. Li, and M. Yin. An exact algorithm with new upper bounds for the maximum k-defective clique problem in massive sparse graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 10174–10183, 2022.

14 M. Grbić, A. Kartelj, S. Janković, D. Matić, and V. Filipović. Variable neighborhood search for partitioning sparse biological networks into the maximum edge-weighted k-plexes. *arXiv preprint arXiv:1807.01160*, 2018. arXiv:1807.01160.

15 X. Li, H. Chen, and H. Yan. The identification of protein complexes from weighted ppi networks using a novel core-attachment method. *BMC Bioinformatics*, 11(1):1–13, 2010. doi:10.1186/1471-2105-11-595.

16 Y. Liu, L. Zhou, and Q. Liu. Community detection based on k-plexes in complex networks. *Physica A: Statistical Mechanics and its Applications*, 503:999–1009, 2018. doi:10.1016/j.physa.2018.02.159.

17 G. Manoussakis. A new decomposition technique for maximal clique enumeration for sparse graphs. *Theoretical Computer Science*, 770:25–33, 2019. doi:10.1016/j.tcs.2018.10.014.

18 C. Verma and A. K. Tripathi. Enumeration of k-defective cliques, 2024. arXiv:2407.16588.

19 Z. Wang, Y. Zhou, M. Xiao, and B. Khoussainov. Listing maximal k-plexes in large real-world graphs. In *Proceedings of the ACM Web Conference 2022*, WWW '22, pages 1517–1527, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3485447.3512198.

20 B. Wu and X. Pei. A parallel algorithm for enumerating all the maximal k-plexes. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 476–483. Springer, 2007. doi:10.1007/978-3-540-71701-0_46.

21 M. Yannakakis. Node- and edge-deletion np-complete problems. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing (STOC)*, pages 253–264. ACM, 1978. doi:10.1145/800133.804355.

22 H. Yu, X. Zhao, and H. Huang. Discovering protein complexes based on k-plexes in protein interaction networks. *PLOS ONE*, 9(7):e101541, 2014. doi:10.1371/journal.pone.0101541.

23 J. Zhou, X. Chen, J. Huang, and Q. Fang. Efficient community detection based on k-plexes in social networks. *Information Sciences*, 512:1172–1192, 2020. doi:10.1016/j.ins.2019.10.057.

24 Y. Zhou, J. Xu, Z. Guo, M. Xiao, and Y. Jin. Enumerating maximal k-plexes with worst-case time guarantee. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 2442–2449, 2020.

Enumeration Kernels for Vertex Cover and Feedback Vertex Set

Marin Bougeret
LIRMM, Université de Montpellier, Montpellier, France

Guilherme C. M. Gomes
LIRMM, Université de Montpellier, CNRS, Montpellier, France
Universidade Federal de Minas Gerais, Belo Horizonte, Brazil

Vinicius F. dos Santos
Universidade Federal de Minas Gerais, Belo Horizonte, Brazil

Ignasi Sau
LIRMM, Université de Montpellier, CNRS, Montpellier, France

Abstract

Enumerative kernelization is a recent and promising area sitting at the intersection of parameterized complexity and enumeration algorithms. Its study began with the paper of Creignou et al. [Theory Comput. Syst., 2017], and development in the area has started to accelerate with the work of Golovach et al. [J. Comput. Syst. Sci., 2022]. The latter introduced polynomial-delay enumeration kernels and applied them in the study of structural parameterizations of the Matching Cut problem and some variants. Few other results, mostly on Longest Path and some generalizations of Matching Cut, have also been developed. However, little success has been seen in enumeration versions of Vertex Cover and Feedback Vertex Set, some of the most studied problems in kernelization. In this paper, we address this shortcoming. Our first result is a polynomial-delay enumeration kernel with $2k$ vertices for Enum Vertex Cover, where we wish to list all solutions with at most k vertices. This is obtained by developing a non-trivial lifting algorithm for the classical crown decomposition reduction rule, and directly improves upon the kernel with $\mathcal{O}(k^2)$ vertices derived from the work of Creignou et al. Our other result is a polynomial-delay enumeration kernel with $\mathcal{O}(k^3)$ vertices and edges for Enum Feedback Vertex Set; the proof is inspired by some ideas of Thomassé [TALG, 2010], but with a weaker bound on the kernel size due to difficulties in applying the q-expansion technique.

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

Keywords and phrases Kernelization, Enumeration, Vertex cover, Crown decomposition, Feedback vertex set

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.23

Related Version *Full Version*: `https://arxiv.org/abs/2509.08475`

Funding *Guilherme C. M. Gomes*: Funded by the European Union, project PACKENUM, grant number 101109317. Views and opinions expressed are however those of the author only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.
Vinicius F. dos Santos: Supported by CNPq (Grants 312069/2021-9, 406036/2021-7, 404479/2023-5, and 308129/2025-3).
Ignasi Sau: French project ELIT (ANR-20-CE48-0008-01).

© Marin Bougeret, Guilherme C. M. Gomes, Vinicius F. dos Santos, and Ignasi Sau;
licensed under Creative Commons License CC-BY 4.0

1 Introduction

Enumeration problems are central objects in both practical and theoretical computer science, with the main example of the former being the branch-and-bound method [35] (and its relatives), universally used in integer programming solvers. Instead of answering a question in either the positive or the negative, in this class of problems the goal is to generate all witnesses, or *solutions*, that satisfy the given problem's constraints, or decide that no solution exists; we denote by $\mathsf{Sol}(x)$ the solution set of an instance x of some fixed problem of interest. Typically, the number of solutions will be exponential in the input size n, and thus *input-sensitive* algorithms, whose running-times are only analyzed with respect to n, are not powerful enough tools to capture all nuances intrinsic to enumeration problems. In particular, *input-polynomial* algorithms, whose running times are of the form $n^{\mathcal{O}(1)}$, are inadequate models of efficient enumeration. Consequently, enumeration algorithms are usually analyzed in the *output-sensitive* context, i.e., their running times depend on both n and the size of the output. In this setting, a popular class of efficiently solvable enumeration problems are those that admit *polynomial-delay* [3, 23, 31] algorithms, where the running time between consecutive outputs must be bounded by $\mathsf{poly}(n)$. These algorithms are unfeasible goals in many interesting cases, as enumeration problems for which the existence of a single solution is an NP-hard problem trivially do not admit such algorithms unless $\mathsf{P} = \mathsf{NP}$.

Orthogonally to enumeration, the theory of parameterized complexity [18, 22, 28] gained significant traction as a means of coping with NP-hardness. In this framework, an instance x is equipped with a *parameter* of interest $k = \kappa(x)$ and the efficient algorithms, known as *fixed-parameter tractable* (FPT), are those of running time $f(k) \cdot n^{\mathcal{O}(1)}$, for some computable function f. An equivalent definition of tractability in this framework, and central to our work, is that of *decision kernelization*: a polynomial-time algorithm, or *kernel*, that compresses an instance (x, k) into an equivalent instance (y, ℓ) such that $|y|, \ell \leq g(k)$ for some computable function g, known as the *size* of the kernel. Beyond their theoretical interest, kernels are also models for theoretically guaranteed *pre-processing* algorithms, a key feature in real-world applications, such as in integer optimization. Polynomial kernels, i.e., when $g(k) = \mathsf{poly}(k)$, are of particular interest, as they typically result in significant reductions in input size. Kernelization theory counts with several robust results and tools [1, 4, 6, 8, 11, 21, 25, 28, 39], including meta-theorems [7, 27, 29, 32], for both upper and lower bounds.

A very successful case study in kernelization is that of Vertex Cover, arguably the most fundamental problem in parameterized complexity. Abu-Khzam et al. [1] presented the first kernel with $\mathcal{O}(k)$ vertices for the problem when parameterized by the solution size k, using the then-novel concept of *crown decompositions* to improve on the previously best known kernel with $\mathcal{O}(k^2)$ vertices, implied by a result of Buss and Goldsmith [13]. Currently, the best known kernels for Vertex Cover have size $2k$ [15], and are based on the half-integral relaxation of Nemhauser and Trotter [38]. Alongside Vertex Cover, the Feedback Vertex Set problem has been quite influential in kernelization theory [9, 12, 39]. When parameterized by the solution size k, it was first shown to admit a kernel with $\mathcal{O}(k^{11})$ vertices by Burrage et al. [12]; this was later improved to a cubic kernel by Bodlaender and van Dijk [9]. Shortly afterwards, Thomassé [39] presented a kernel with $\mathcal{O}(k^2)$ vertices and edges by introducing a generalization of crown decompositions known as *q-expansions*, which have since been used in other examples in the literature [25, 26, 33, 34]. Thomassé's kernel size has also been shown to be asymptotically optimal by Dell and Marx [21].

Over the last years, there have been mounting efforts to extend parameterized complexity to the enumeration setting [5, 14, 17, 19, 20, 24, 30, 33, 36, 37], with *input*-FPT and FPT-*delay* having their expected definitions; the work of Creignou et al. [17] has been particularly influential in this endeavor, and we refer to it for a broader discussion on parameterized enumeration. We will also not further discuss input-sensitive enumeration, focusing on delay-based algorithms. While parameterized enumeration algorithms have been developed for some decades, it took much longer, however, for the area of *enumerative kernelization* to emerge, even though kernelization itself did play a significant role in several works [17, 19, 24].

Only very recently Golovach et al. [30] defined *polynomial-delay enumeration (PDE) kernels*, which are equivalent to the existence of FPT-delay algorithms, much like FPT algorithms and kernels are equivalent in the decision world; previous models, such as full kernels [19] and enum-kernels [17], had issues that made them unsatisfactory definitions for enumerative kernelization. Intuitively, a PDE kernel has two parts: the first, $\mathcal{A}_1$, is the *compression algorithm* and has the same constraints as a decision kernel, while the second $\mathcal{A}_2$, known as the *lifting algorithm*, receives the input (x, k), the output (y, ℓ) of $\mathcal{A}_1$, and a solution $Y \in \mathsf{Sol}(y)$, and must output, with $\mathsf{poly}(|x| + |y| + k + \ell)$-delay, a non-empty subset $S_Y \subseteq \mathsf{Sol}(x)$; additionally, the S_Y's must partition $\mathsf{Sol}(x)$. PDE kernels have been applied to some problems [14, 30, 33, 36], mainly restricted to variants of the Matching Cut problem; [33] is the unique exception, as it investigates an enumeration variant of the Longest Path problem; we remark that they apply the q-expansion technique to obtain their results. Despite being stated as an enum-kernel, Creignou et al.'s [17] $\mathcal{O}(k^2)$ kernel for Enum Vertex Cover is a PDE kernel. Despite these positive results, the most basic techniques of decision kernelization theory seem quite difficult to adapt to the enumeration setting. Motivated by this phenomenon, we examine the best known kernels for Vertex Cover and Feedback Vertex Set and seek to adapt their ideas and techniques to their respective enumeration variants.

Our contributions. Our first result is a PDE kernel for Enum Vertex Cover with $2k$ vertices, and directly improves upon the $\mathcal{O}(k^2)$ kernel of Creignou et al. [17]; the problem is formally defined as follows.

Enum Vertex Cover
Instance: A graph G and an integer k.
Enumerate: All vertex covers of G of size at most k.

Our proof is a direct generalization of the kernel with $2k$ vertices for Vertex Cover; this, in turn, is based on the celebrated crown decomposition technique. Specifically, we obtain the following theorem.

▶ **Theorem 1.** Enum Vertex Cover *admits a PDE kernel with at most* $2k$ *vertices when parameterized by the maximum desired size of a solution* k*.*

Intuitively, our kernel works by applying the classic crown reduction that deletes the head H and crown C of the decomposition, reduces k by $|H|$, and returns only the body B. The $2k$ bound is consequence of the Nemhauser-Trotter decomposition theorem [38] and the fact that it either yields a crown decomposition or gives a bound on the size of the input graph. The complicated part is showing how to design a lifting algorithm; we do so by proving that Enum Vertex Cover restricted to *crowned graphs*, i.e., graphs that can be partitioned into $H \cup C$ where C is an independent set and there is an H-saturating matching between H and C, admits a polynomial-delay algorithm. Given a solution S of

the reduced instance, our lifting algorithm has three phases: (i) brute force how the *slack* $k - (|S| + |N(B \setminus S) \cap H|)$ is distributed in C, (ii) compute sets of vertices forced to be added to and forbidden in the solution given the output of (i), and (iii) solve the instance $(G', |V(G')|/2)$ induced by the remaining vertices. The key observations are that: (a) every configuration tested in (i) leads to a solution of the input, and (b) G' is a crowned graph with a perfect matching.

We then proceed to show a cubic PDE kernel for the Enum Feedback Vertex Set problem, formally defined as below.

Enum Feedback Vertex Set
Instance: A multigraph G and an integer k.
Enumerate: Every feedback vertex set of G of size at most k.

We follow a similar direction to Thomassé's quadratic kernel [39]: we first design reduction rules to lower bound the minimum degree of the instance; then, we show that upper bounding the maximum degree is enough to lead to a kernel; and finally, we present reduction rules that yield said upper bound. The core ingredient of Thomassé's proof – the computation of a 2-expansion in an auxiliary graph – seems to fail in the enumeration setting, and we are unable to generalize it. Nevertheless, we obtain Theorem 2. Our set of reduction rules is mostly different, which leads to a quadratic instead of a linear bound on the maximum degree. As our lower bound on the minimum degree is also different, we must prove a slight generalization on the bound used in Thomassé's proof.

▶ **Theorem 2.** *There is a PDE kernel for* Enum Feedback Vertex Set *parameterized by the size of the solution k with $\mathcal{O}(k^3)$ vertices.*

Organization. In Section 2 we present some basic preliminaries about graphs, parameterized complexity, and enumeration problems. In Section 3, we present our linear PDE kernel for Enum Vertex Cover. In Section 4, we present our cubic PDE kernel for Enum Feedback Vertex Set and discuss some challenges in using rules based on q-expansions. We present concluding remarks and possible research directions in Section 5. Due to space limitations, the proofs of the statements marked with "$(\star)$" have been omitted and can be found in the full version available online (`https://arxiv.org/abs/2509.08475`).

2 Preliminaries

We denote the set $\{1, 2, \ldots, n\}$ by $[n]$. We use standard graph-theoretic notation, and we consider simple undirected graphs without loops or multiple edges; see [10] for any undefined terminology. When the graph is clear from the context, the degree (that is, the number of neighbors) of a vertex v is denoted by $\deg(v)$, and the number of neighbors of a vertex v in a set $A \subseteq V(G)$ and its neighborhood in it are, respectively, denoted by $\deg_A(v)$ and $N_A(v)$; we also define $N(S) = \bigcup_{v \in S} N(v) \setminus S$. A *matching* M of a graph G is a subset of edges of G such that no vertex of G is incident to more than one edge in M; for simplicity, we define $V(M) = \bigcup_{uv \in M} \{u, v\}$ and refer to it as the set of *M-saturated vertices*. The *subgraph of G induced by X* is defined as $G[X] = (X, \{uv \in E(G) \mid u, v \in X\})$. A *feedback vertex set* $X \subseteq V(G)$ is such that $G \setminus X \coloneqq G[V(G) \setminus X]$ is a forest; the *feedback vertex number* of G is the size of a feedback vertex set of minimum size. Given adjacent $u, v \in V(G)$, a *contraction of u into v* is an operation that outputs the graph $G \setminus \{u\}$ with the added edges $\{vw \mid w \in N_G(u) \setminus \{v\}\}$. An algorithm is said to run with *polynomial delay* [16] if the

times before the first output (called the *precalculation period*), after the last output (*the postcalculation period)*, and in between two consecutive outputs are bounded by a polynomial in the input size only.

We refer the reader to [18, 22] for basic background on parameterized complexity and present here only definitions pertaining to parameterized enumeration.

▶ **Definition 3** (Parameterized enumeration problem (Creignou et al. [17], Golovach et al. [30]))**.** *A* parameterized enumeration problem *over a finite alphabet* Σ *is a tuple* $\Pi = (L, \mathsf{Sol}, \kappa)$*, shorthanded by* Π_κ*, such that:*

i. $L \subseteq \Sigma^\star$ *is a decidable language;*
ii. $\mathsf{Sol} : \Sigma^\star \mapsto 2^{\Sigma^\star}$ *is a computable function such that, for every* instance $x \in \Sigma^\star$*,* $\mathsf{Sol}(x)$ *is finite and we have* $\mathsf{Sol}(x) \neq \emptyset$ *if and only if* $x \in L$*;*
iii. $\kappa : \Sigma^\star \mapsto \mathbb{N}$ *is the* parameterization*.*

An instance x *of* Π_κ *is a* no-instance *if* $\mathsf{Sol}(x) = \emptyset$*, and is a* yes-instance *otherwise.*

▶ **Definition 4** (Polynomial-delay enumeration kernel)**.** *Let* Π_κ *be a parameterized enumeration problem. A* polynomial-delay enumeration kernel, PDE kernel *for short, is a pair of algorithms* $\mathcal{A}_1, \mathcal{A}_2$ *such that, given an instance* x *of* Π_κ *and* $\kappa(x)$*:*

- *The* compression algorithm $\mathcal{A}_1$ *runs in* $\mathsf{poly}(|x| + \kappa(x))$*-time and outputs an instance* y *of* Π_κ *satisfying* $\kappa(y), |y| \leq f(\kappa(x))$*, with* f *being a computable function, and with* $\mathsf{Sol}(y) \neq \emptyset$ *if and only if* $\mathsf{Sol}(x) \neq \emptyset$*;*
- *The* lifting algorithm $\mathcal{A}_2$ *receives* x*,* $y = \mathcal{A}_1(x, \kappa(x))$*,* $\kappa(x)$*,* $\kappa(y)$*, some* $Y \in \mathsf{Sol}(y)$*, and outputs, with polynomial delay on its input, a non-empty* $S_Y \subseteq \mathsf{Sol}(x)$*. Moreover,* $\{S_Y \mid Y \in \mathsf{Sol}(y)\}$ *is a partition of* $\mathsf{Sol}(x)$*.*

A *graph subset problem* is a problem with a graph as input and a solution being a subset of its vertices. We say that a PDE kernel for a graph subset problem is *extension-only* if, given a solution $Y \in \mathsf{Sol}(y)$, the lifting algorithm works by only adding some vertices in $x \setminus y$ to Y. Intuitively, this constraint implies that a solution of the input instance x cannot be generated by two different solutions of the compressed instance y, so it only remains to prove that every solution of x is generated from some solution of y and every solution of y leads to some solution of x. Both kernels developed in this work are extension-only. We remark that the kernels for Enum Matching Cut and its variants developed by Golovach et al. [30] are not extension only, as their lifting algorithms remove some vertices of the compressed instance's solutions.

3 A kernel with $2k$ vertices for Enum Vertex Cover

In this section we prove Theorem 1. Our main tools for this result are crowned graphs and crown decompositions, given in Definition 5.

▶ **Definition 5.** *A graph is a* crowned graph *if its vertex set can be partitioned into* $H \cup C$*, such that* C *is an independent set, and such that there is a matching* M *between* H *and* C *where* $|M| = |H|$*. The set* C *is called the* crown*, and* H *is called the* head*. Given a graph* G*, a* crown decomposition *[18] of width* t *is a partition* (C, H, B) *of* $V(G)$ *such that* $G[H \cup C]$ *is a crowned graph (with head* H*),* $N_G(C) = H$*, and* $|H| = t$*. Set* B *is called the* body*.*

To that end, we first reduce the task of finding a PDE kernel to an enumeration problem Enum Crown, and then reduce Enum Crown to an even more structured enumeration problem Enum Small Crown for which we provide a polynomial-delay enumeration algorithm. Let us first define our problem.

Enum Crown
Instance: A crowned graph $G = (H \cup C, E)$ and an integer x in $[0, \lvert C\rvert]$.
Enumerate: All vertex covers of G of size exactly $\lvert H\rvert + x$.

The problem Enum Small Crown corresponds to the special case of Enum Crown where $|C| = |H|$ and $x = 0$. Given an input (G, x) of Enum Crown, we denote by $\mathsf{Sol}(G, x)$ the set of solutions. In the same way, we define $\mathsf{Sol}(G)$ for Enum Small Crown.

▶ Remark 6. Every instance of Enum Crown has a solution that takes the head and any x vertices in the crown. If M is an H-saturating matching between H and C, then every solution S of Enum Small Crown and every $e \in M$ satisfy $|S \cap V(e)| = 1$, where $V(e)$ denotes the endpoints of e.

Our kernelization algorithm consists in the exhaustive application of Rules VC.1 and VC.2, the latter of which can be applied in polynomial time due to Lemma 7.

▶ **Reduction Rule VC.1.** *Let (G, k) be the input instance of* Enum Vertex Cover. *If v is an isolated vertex of G, remove v from G and set $k \leftarrow k$.*

Proof of safeness and lifting algorithm of Rule VC.1. Let $G' = G \setminus \{v\}$. The forward direction is immediate; $S \in \mathsf{Sol}(G, k)$ if and only if $S \setminus \{v\} \in \mathsf{Sol}(G', k)$. For the converse direction, it is also immediate that $S' \in \mathsf{Sol}(G', k)$ if and only if $S' \in \mathsf{Sol}(G, k)$; however, observe that $(S' \cup \{v\}) \in \mathsf{Sol}(G, k)$ if and only if $|S'| < k$ as v is isolated, and we must also output $(S' \cup \{v\})$ if possible. ◀

As pointed out by Chlebík and Chlebíkova [15], the Nemhauser-Trotter [38] decomposition based on the half-integral relaxation of Vertex Cover is a crown decomposition.

▶ **Lemma 7** ([38] and [15])**.** *Let G be a graph without isolated vertices and with at least $2k + 1$ vertices. Then, there is a polynomial-time algorithm that either:*
- *decides that no half-integral vertex cover of weight at most k exists; or*
- *finds a crown decomposition of G of width at most k.*

▶ **Reduction Rule VC.2** (Crown reduction)**.** *Let (G, k) be an instance of* Enum Vertex Cover *with (C, H, B) a crown decomposition of G of width at most k. Set $G' \leftarrow G \setminus (H \cup C)$, $k' \leftarrow k - |H|$, and return (G', k').*

▶ **Lemma 8.** *If* Enum Crown *has a polynomial-delay enumeration algorithm $\mathcal{A}$, then Rule VC.2 is a reduction rule with an extension-only lifting algorithm.*

Proof. Let us show how to construct the lifting algorithm $\mathcal{R}_b$ using $\mathcal{A}$. Given a solution S' of the reduced instance (G', k'), let $x = k' - |S'|$ (where $x \geq 0$) be the *slack* of the solution S', and $F = N(B \setminus S') \cap H$ be the set of *forced* vertices in H, i.e. vertices that are adjacent to some vertex of B not picked in S'. Let $H' = H \setminus F$. For any $\ell \in [0, \min(|C|, x)]$, define $x_\ell = |H'| + \ell$, and observe that $(G[H' \cup C], x_\ell)$ is a valid instance of Enum Crown as $x_\ell \leq |C|$. For any $\ell \in [0, \min(|C|, x)]$, $\mathcal{R}_b$ outputs all solutions of the form $S' \cup F \cup S_\ell$, where $S_\ell \in \mathsf{Sol}(G[H' \cup C], x_\ell)$ are enumerated by $\mathcal{A}$. This completes the description of $\mathcal{R}_b$, so let us now prove that it adheres to the requirements of Definition 4. The first item is satisfied as it is well known that the crown reduction is safe for the decision version of the problem, in the sense that (G, k) is a yes-instance if and only if (G', k') is. Let us now check the second item, more precisely:
- For every $S' \in \mathsf{Sol}(G', k')$, the set $\mathcal{R}_b(G, G', k, k', S')$ is non-empty, and can be enumerated with polynomial-delay; and
- $\{\mathcal{R}_b(G, G', k, k', S') \mid S' \in \mathsf{Sol}(G', k')\}$ is a partition of $\mathsf{Sol}(G, k)$.

Let us start with the first property. Notice that there exists at least one $\ell \in [0, \min(|C|, x)]$, and as for any such ℓ, tuple $((H', C), x_\ell)$ is an instance of Enum Crown (which always has a solution by Remark 6), we get that $\mathcal{R}_b(G, G', k, k', S') \neq \emptyset$. Moreover, $\mathcal{R}_b(G, G', k, k', S')$ runs with polynomial-delay, as for each ℓ, $\mathcal{A}$ enumerates $\mathsf{Sol}(G[H' \cup C], x_\ell)$ with polynomial-delay, and there is no additional verification required in $\mathcal{R}_b$ to avoid duplicate solutions of the form $S' \cup F \cup S_\ell$ generated by different values of ℓ, as for any ℓ these solutions have size exactly $|S'| + |F| + x_\ell$.

Let us now consider the second property. Observe first that for any $S' \in \mathsf{Sol}(G', k')$, any $S \in \mathcal{R}_b(G, G', k, k', S')$ is indeed a solution of (G, k) as $V(G)$ can be partitioned into $V_1 = V(G') \cup F$ and $V_2 = H' \cup C$, and $S = S' \cup F \cup S_\ell$ where $S' \cup F$ is a vertex cover of $G[V_1]$, S_ℓ is a vertex cover of $G[V_2]$, and there is no edge between $V_1 \setminus S$ and V_2. Let us now prove that $\mathsf{Sol}(G, k) \subseteq \bigcup_{S' \in \mathsf{Sol}(G', k')} \mathcal{R}_b(G, G', k, k', S')$. Let $S \in \mathsf{Sol}(G, k)$ and $S' = S \setminus (H \cup C)$. As there is a matching of size $|H|$ between the head and the crown, $|S \cap (H \cup C)| \geq |H|$ and thus $S' \in \mathsf{Sol}(G', k')$. Now, let us prove that $S \in \mathcal{R}_b(G, G', k, k', S')$. Observe first that as S necessarily contains F, S is partitioned into $S = S' \cup F \cup (S \cap (H' \cup C))$, where $S \cap (H' \cup C)$ is a vertex cover of $G[H' \cup C]$. Let ℓ_0 be such that $|S \cap (H' \cup C)| = |H'| + \ell_0$. It remains to prove that $\ell_0 \in [0, \min(|C|, x)]$, as this implies that $(S \cap (H' \cup C)) \in \mathsf{Sol}(G[H' \cup C], x_{\ell_0})$, and thus that $S \in \mathcal{R}_b(G, G', k, k', S')$. First, we have $\ell_0 \geq 0$ as $(C, H', \emptyset)$ remains a crown decomposition, and thus there is a matching saturating H' in $G[H' \cup C]$. Let $x = k' - |S'|$ where $x \geq 0$. Moreover, $|S| = |S'| + |F| + |(S \cap (H' \cup C))| = k' - x + |F| + |H'| + \ell_0 = k - x + \ell_0 \leq k$, implying that $\ell_0 \leq x$. As $\ell_0 \leq |C|$ by definition, we get the claimed inequality. Finally, observe that $S \cap V(G') = S'$, so it follows that $\mathcal{R}_b$ is an extension-only lifting algorithm, which implies that, for any two different solutions S'_1, S'_2 of $\mathsf{Sol}(G', k')$, the sets $\mathcal{R}_b(G, G', k, k', S'_1)$, and $\mathcal{R}_b(G, G', k, k', S'_2)$ are disjoint. ◀

With this conditional result, we are now able to state a conditional kernelization lemma.

▶ **Lemma 9** (⋆). *If* Enum Crown *has a polynomial-delay enumeration algorithm, then* Enum Vertex Cover *admits a PDE kernel (with extension-only lifting algorithm) with at most* $2k$ *vertices.*

The remainder of this section is dedicated to proving that Enum Crown admits a polynomial-delay enumeration algorithm. In what follows, given a matching M, and a subset $T \subseteq V(M)$ such that for any $e \in M$, $|V(e) \cap T| \leq 1$, we define $M_V(T) \coloneqq \{v' \mid vv' \in M \text{ and } v \in T\}$ and call it the *image* of T by M. We need the following procedure and its associated technical lemma to obtain our lifting algorithm. Intuitively, in a small crowned graph G, exactly one endpoint of each edge of an H-saturating matching M can be in a vertex cover; $\mathsf{propagate}_{\mathsf{dir}}(G, M, X_0)$ decides which vertices must be included and which must be excluded from a solution if we fix X_0 to be part of the it.

▶ **Definition 10** ($\mathsf{propagate}_{\mathsf{dir}}(G, M, X_0)$). *Given a small crowned graph $G = (H \cup C, E)$ (meaning with $|H| = |C|$) and $X_0 \subseteq H$, the procedure $\mathsf{propagate}_{\mathsf{dir}}(G, M, X_0)$ outputs two sets $(F, \bar{F})$ as follows (see Figure 1). Let $\vec{G} = (A \cup B, \vec{E})$ be an oriented bipartite graph with $A = H$, $B = C$, where:*

- *for any edge e between A and B with $e \in M$, we add its oriented version from A to B.*
- *for any edge e between A and B with $e \notin M$, we add its oriented version from B to A.*

Let $Z \subseteq A \cup B$ be the vertices that can be reached from X_0 in $\vec{G}$, including X_0 itself. Then, $\mathsf{propagate}_{\mathsf{dir}}(G, M, X_0)$ outputs $(F, \bar{F})$ where $F = Z \cap A$ and $\bar{F} = Z \cap B$.

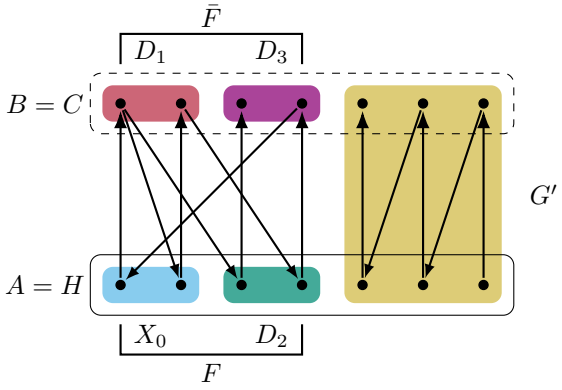


Figure 1 Example of an application of $\mathsf{propagate}_{\mathsf{dir}}(G, M, X_0)$ that stopped at distance three from X_0, with the graph depicted corresponding to $\vec{G}$. Edges of M correspond to the vertical arcs, while sloped arcs correspond to all other arcs of $\vec{G}$. The light-blue-shaded vertices correspond to X_0, the red-shaded to D_1 (the vertices at distance one from X_0), the teal-shaded to D_2, the purple-shaded to D_3, and the yellow-shaded vertices correspond to the graph $G' = G[(H \setminus F) \cup (C \setminus \bar{F})]$.

▶ **Lemma 11.** *Let G be a small crowned graph, M a perfect matching of G between H and C, $X_0 \subseteq H$, and $(F, \bar{F}) = \mathsf{propagate}_{\mathsf{dir}}(G, M, X_0)$. Then:*

1. *$\bar{F} = M_V(F)$ (where $F \subseteq H$ and $\bar{F} \subseteq C$) and for any solution $S \in \mathsf{Sol}(G)$ such that $X_0 \subseteq S$, S contains F and avoids $\bar{F}$.*
2. *For any solution $S \in \mathsf{Sol}(G)$ such that $X_0 \subseteq S$, if we define $H' = H \setminus F$ and $C' = C \setminus \bar{F}$),then $G' = G[H' \cup C']$ is a small crowned graph and $S \cap (C' \cup H') \in \mathsf{Sol}(G')$.*
3. *For any solution $S' \in \mathsf{Sol}(G')$, $F \cup S'$ is a solution of G (that contains X_0).*

Proof. Let $\vec{G} = (A \cup B, \vec{E})$ be the oriented bipartite graph defined in Definition 10.

Proof of Item 1. Let $S \in \mathsf{Sol}(G)$ such that $X_0 \subseteq S$. For any $i \geq 0$, let D_i be the vertices of (A, B) at distance i from X_0 in $\vec{G}$ (e.g. $D_0 = X_0$, $D_1 = M_V(X_0)$, see Figure 1). Let us show by induction that for any i, if i is even, then $D_i \subseteq A \cap S$, and if i is odd then $D_i \subseteq B \setminus S$, and $D_i = M_V(D_{i-1})$. Observe that the statement holds for $i = 0$. Let us now assume that it holds for $i - 1$ and consider i. If i is odd, observe first that $D_i = M_V(D_{i-1})$ by the definition of $\vec{G}$, and, by Remark 6, for any edge $v, v' \in M$ with $v \in D_{i-1}$ and $v' \in D_i$, $v' \notin S$. If i is even, as $D_{i-1} \cap S = \emptyset$ and S is a vertex cover of G, then $D_i \subseteq S$, and, as $\vec{G}$ is bipartite, we also get that $D_i \subseteq A$. Now, as $F = \bigcup_{i \equiv 0 \mod 2} D_i$ and $\bar{F} = \bigcup_{i \equiv 1 \mod 2} D_i$, we get $\bar{F} = M_V(F)$, and Item 1 follows.

Proof of Item 2. Observe first that G' is indeed a crowned graph as $\bar{F} = M_V(F)$. Now, as $S \in \mathsf{Sol}(G)$ we get $|S| = |H|$; by Remark 6 we get $|S \cap (F \cup \bar{F})| = |F|$, leading to $|S \cap (C' \cup H')| = |H'|$.

Proof of Item 3. Let $S' \in \mathsf{Sol}(G')$, and let us consider $S = F \cup S'$. Observe that $H \cup C$ is partitioned into $F \cup \bar{F}$ and $H' \cup C'$, that F is a vertex cover of $G[F \cup \bar{F}]$ (as $\bar{F} = M_V(F)$), and S' is a vertex cover of $G[H' \cup C']$. By definition of F and $\bar{F}$, there is no edge between $\bar{F}$ and $H' \cup C'$, implying that S is a vertex cover of $G[H \cup C]$. Since $|S| = |H|$, we get that S is a solution of G (that contains X_0 by definition of F). ◀

▶ **Lemma 12.** *If* Enum Small Crown *admits a polynomial-delay algorithm, then* Enum Crown *also admits such an algorithm.*

Proof. Let us define a polynomial-delay enumeration algorithm $\mathcal{A}$ for Enum Crown. Let (G, x) be an instance of Enum Crown with $V(G) = H \cup C$, and let M be a matching from H to C that saturates H. Given $T \subseteq V(G)$, we define $M(T) = \{e \in M \mid V(e) \subseteq T\}$, i.e., the set of edges of M entirely in T. For any $S \in \mathsf{Sol}(G, x)$, the *signature of* S is composed of two sets, $C_1(S) = V(M(S)) \cap C$ and $C_2(S) = (C \setminus V(M)) \cap S)$. Observe that

- $|C_1(S)| = |M(S)|$, where $|C_1(S)| \leq x$ (as $|C_1(S)| + |H| \leq |S| = |H| + x$);
- $x - (|C| - |M|) \leq |C_1(S)|$ (as if $x - (|C| - |M|) > |C_1(S)|$ then as $C_2(S) \subseteq (C \setminus V(M))$, we would have $|S| = |H| + |C_1(S)| + |C_2(S)| < |H| + x$), and $|C_2(S)| = x - |C_1(S)|$;
- By defining $\bar{X}_1(S) = (C \setminus V(M)) \setminus C_2(S)$, we have $S \cap \bar{X}_1(S) = \emptyset$.

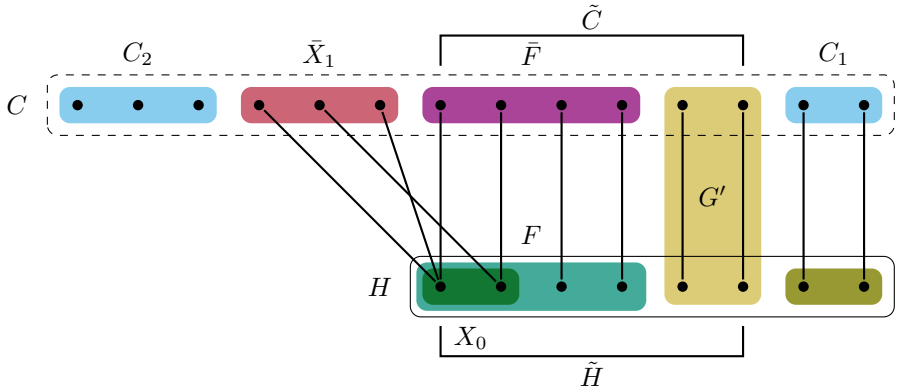


Figure 2 Example of the structures involved in the algorithm $\mathsf{propagate}_{\mathsf{large}}$, used in the proof of Lemma 12. The five light-blue-shaded vertices correspond to the inputs C_1, C_2 of $\mathsf{propagate}_{\mathsf{large}}$; the two olive-shaded vertices are forced to be in the solution by our choice of C_1. $\bar{X}_1$, i.e., the set of vertices of $C \setminus V(M)$ that should be forbidden in a solution to G, is represented by the three red-shaded vertices; the dark-green shaded vertices are forced to be in every solution that avoids $\bar{X}_1$, and, together with $\tilde{G} = G[\tilde{H} \cup \tilde{C}]$, composes the input to the call to $\mathsf{propagate}_{\mathsf{dir}}$ used in $\mathsf{propagate}_{\mathsf{large}}$; the four teal-shaded and four purple-shaded vertices are the outputs $F, \bar{F}$ of $\mathsf{propagate}_{\mathsf{dir}}$, and, together with $\bar{X}_1$, make up the output of $\mathsf{propagate}_{\mathsf{large}}$. Finally, the four yellow-shaded vertices represent G', the instance of Enum Small Crown to which $\mathcal{A}_s$ will be applied.

Our next step is guessing which edges of the H-saturating matching M will have both endpoints in the solution S; in the next propagation procedure, this is represented by set C_1, i.e. which vertices of $C \cap V(M)$ must be accompanied by their image in S. We also guess which vertices of C that *are not* matched to H will be included in a solution; this is represented by C_2. Finally, the choice of C_1 and C_2 implies that the vertices $\bar{X} \subseteq C \setminus (V(M) \cup C_2)$ are incident to yet uncovered edges. Formally, for any $d \in [\max(0, x - (|C| - |V(M)|)), x]$, any set $C_1 \subseteq V(M) \cap C$ of size d, and any set $C_2 \subseteq C \setminus V(M)$ of size $x - d$, we define (see Figure 2) $\mathsf{propagate}_{\mathsf{large}}(C_1, C_2)$ that computes $(F, \bar{F}, \bar{X}_1)$ where $\bar{X}_1 = (C \setminus V(M)) \setminus C_2$, and $(F, \bar{F}) = \mathsf{propagate}_{\mathsf{dir}}(\tilde{G}, X_0)$ with $X_0 = N(\bar{X}_1)$, and $\tilde{G} = G[\tilde{H} \cup \tilde{C}]$, where $\tilde{H} = H \setminus M_V(C_1)$, and $\tilde{C} = M_V(\tilde{H})$.

▷ Claim 13. For any $S \in \mathsf{Sol}(G, x)$ of signature (C_1, C_2),

1. S contains F and avoids $\bar{F} \cup \bar{X}_1$;
2. By defining $H' = \tilde{H} \setminus F$ and $C' = \tilde{C} \setminus \bar{F}$, $G' = G[H' \cup C']$ is a small crowned graph and $(S \cap (C' \cup H')) \in \mathsf{Sol}(G')$.

Proof. Let $S \in \mathsf{Sol}(G,x)$ have signature (C_1,C_2). By definition of $C_2(S)$, we get that $S \cap \bar{X}_1 = \emptyset$, implying also that $X_0 \subseteq S$. Moreover, by definition of $C_1(S)$, no edge $e \in M$ with $V(e) \subseteq \tilde{H} \cup \tilde{C}$ has both its endpoints in S. Thus, if we define $\tilde{S} = S \cap (\tilde{H} \cup \tilde{C})$, we get that $\tilde{S} \in \mathsf{Sol}(\tilde{G})$ and that $X_0 \subseteq \tilde{S}$. By Lemma 11, we obtain the two claimed properties. ◁

We are now ready to define $\mathcal{A}$. To this end, let $\mathcal{A}_s$ be the polynomial-delay algorithm for Enum Small Crown. For any $d \in [\max(0, x-(|C|-|V(M)|)), x]$, any set $C_1 \subseteq V(M) \cap C$ of size d, and any set $C_2 \subseteq C \setminus V(M)$ of size $x-d$, $\mathcal{A}$ runs $\mathsf{propagate}_{\mathsf{large}}(C_1,C_2)$ that computes $(F,\bar{F},\bar{X}_1)$, and defines G', H', and C' as in Item 2 of Claim 13. Note that G' may be the empty graph. Then, $\mathcal{A}$ outputs all solutions of the form $C_1 \cup M(C_1) \cup C_2 \cup F \cup S'$, where $S' \in \mathsf{Sol}(G')$ are enumerated by $\mathcal{A}_s$. Observe that we can indeed call $\mathcal{A}_s$ on G' as according to Item 2, G' is a small crowned graph.

Let us first prove that there is no repetition. Observe that for each d, C_1, C_2 considered by $\mathcal{A}$, all enumerated solutions of the form $C_1 \cup M(C_1) \cup C_2 \cup F \cup S'$ (where $S' \in \mathsf{Sol}(G')$) have a signature (C_1,C_2). Now, it is sufficient to observe that two solutions of (G,x) with different signatures are necessarily different, which implies that there is no repetition.

Let us now prove that all enumerated elements are in $\mathsf{Sol}(G,x)$. Let $S = C_1 \cup M(C_1) \cup C_2 \cup F \cup S'$, where $S' \in \mathsf{Sol}(G')$ exists (as $S' = S \cap V(G')$ and G' is an induced subgraph of G) and is enumerated by $\mathcal{A}$. By Item 3 of Lemma 11, $(S' \cup F) \in \mathsf{Sol}(\tilde{G})$ and contains X_0. Now, observe that $H \cup C$ is partitioned into $V_1 = C_1 \cup M(C_1) \cup C_2 \cup \bar{X}_1$, and $V_2 = \tilde{H} \cup \tilde{C}$. Notice that $S \cap V_i$ is a vertex cover of $G[V_i]$ for $i \in \{1,2\}$. Moreover, as $V_1 \setminus S = \bar{X}_1$, the only edges between V_1 and V_2 not covered by $S \cap V_1$ are edges between $\bar{X}_1$ and $N(\bar{X}_1) = X_0$, but as $S \cap V_2 \supseteq X_0$, S is indeed a vertex cover of G. Finally, $|S| = 2|C_1| + |C_2| + |F| + |H'| = 2|C_1| + |C_2| + |\tilde{H}| = |H| + |C_1| + |C_2| = |H| + x$, and so we have $S \in \mathsf{Sol}(G,x)$.

Towards proving that all elements of $\mathsf{Sol}(G,x)$ are enumerated, let $S \in \mathsf{Sol}(G,x)$, and $(C_1(S),C_2(S))$ be its signature. By definition of $\mathcal{A}$, there exists C_1 and C_2 considered by $\mathcal{A}$ such that $C_i = C_i(S)$ for $i \in \{1,2\}$. Let $(F,\bar{F},\bar{X}_1) = \mathsf{propagate}_{\mathsf{large}}(C_1,C_2)$. By Item 1, $S \supseteq F$, and $S \cap (\bar{F} \cup \bar{X}_1) = \emptyset$, which implies that $S = C_1 \cup M(C_1) \cup C_2 \cup F \cup S'$. As $S \cap (C' \cup H') = S'$, Item 2 implies that $S' \in \mathsf{Sol}(G')$, and so S' is enumerated by $\mathcal{A}_s$.

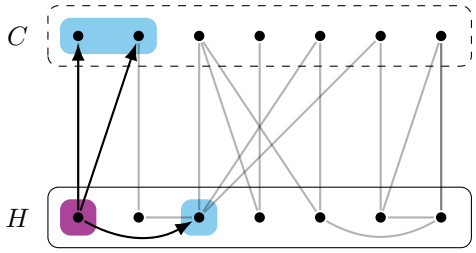
It remains to prove that there is only polynomial-time between two consecutive outputs of $\mathcal{A}$. Notice first that $\mathsf{propagate}_{\mathsf{large}}$ runs in polynomial time. Then, for each fixed C_1, C_2 considered by $\mathcal{A}$, as $\mathcal{A}_s$ runs in polynomial-delay, there is at most polynomial-time between two consecutive solutions having the same signature (C_1,C_2). Moreover, and as for any choice of C_1, C_2 considered by $\mathcal{A}$, we know that $\mathsf{Sol}(G') \neq \emptyset$ (by Remark 6), there is also a polynomial-delay between two consecutive solutions with different signatures. ◀

The proof of Lemma 16 requires a more involved variant of the propagation algorithm, where the goal is to avoid a prescribed vertex v.

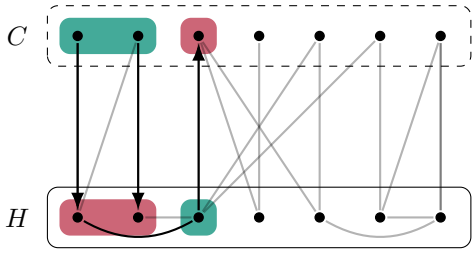
▶ **Definition 14** ($\mathsf{propagate}_{\mathsf{all}}(G,v)$)**.** *Given a small crowned graph $G = (H \cup C, E)$ (i.e., $|H| = |C|$), and $v \in H$, the procedure* $\mathsf{propagate}_{\mathsf{all}}(G,v)$ *either* fails*, or outputs two sets $(F,\bar{F})$ that, intuitively, are forced to be included or avoided in a solution that avoids v and, upon removal, will yield a small crowned graph. We recursively define them as follows. Let $\bar{F}_0 = \{v\}$. Observe that $N(\bar{F}_0)$ intersects C (because of the perfect matching M between H and C, and may intersect H). Now, for any $i \geq 1$ (see Figure 3):*

- *to compute F_i: if there exists $e \in M$ such that $V(e) \subseteq N(\bar{F}_{i-1})$, we fail, and otherwise we define $F_i = N(\bar{F}_{i-1})$.*
- *to compute $\bar{F}_i$: if $M(F_i)$ is not an independent set, we fail, and otherwise we define $\bar{F}_i = M(F_i)$; note that $\bar{F}_{i-1} \subseteq \bar{F}_i$.*

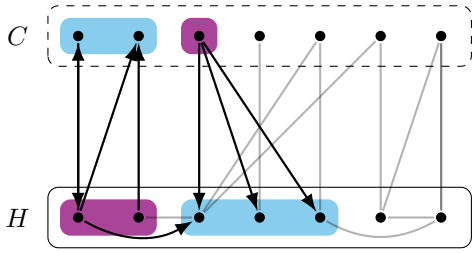
We continue this process until it fails or until $F_i = F_{i-1}$ and $\bar{F}_i = \bar{F}_{i-1}$. If there is no failure, $\mathsf{propagate}_{\mathsf{all}}(G,v)$ *returns $(F,\bar{F}) = (F_i,\bar{F}_i)$.*



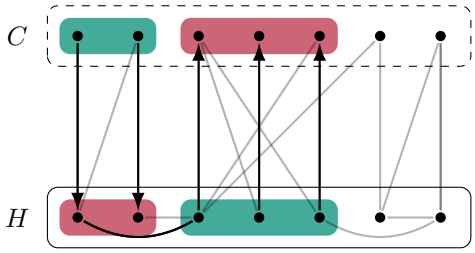
(a) Computing F_1 (cyan) from $\bar{F}_0$ (purple).



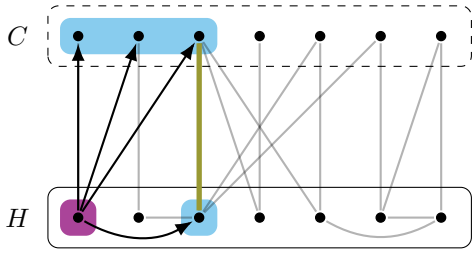
(b) Computing $\bar{F}_1$ (red) from F_1 (teal).



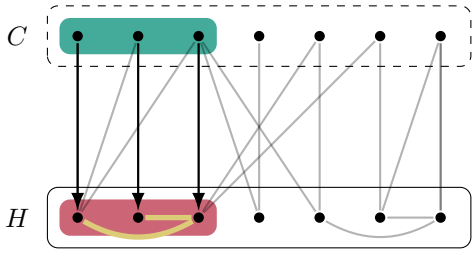
(c) Computing F_2 (cyan) from $\bar{F}_1$ (purple).



(d) Computing $\bar{F}_2$ (red) from F_2 (teal).



(e) Failing to compute F_1 (cyan) from $\bar{F}_0$ (purple).



(f) Failing to compute $\bar{F}_1$ (red) from F_1 (teal).

Figure 3 Examples of executions of algorithm $\mathsf{propagate}_{\mathsf{all}}$. Vertical edges correspond to M. We reduce the width of the edges of G not involved in the corresponding operation for clarity. An arc (x, y) indicates that vertex y was included in the set currently being built by virtue of the membership of x in the root set. The edges that cause the failures in the computation of F_i or $\bar{F}_i$ are thickened and colored differently.

▶ **Lemma 15.** *Let $G = (H \cup C, E)$ be a small crowned graph, and $v \in H$.*

1. *If $\mathsf{propagate}_{\mathsf{all}}(G, v)$ fails, then there is no $S \in \mathsf{Sol}(H, C)$ such that $v \notin S$.*
2. *Otherwise, if $\mathsf{propagate}_{\mathsf{all}}(G, v)$ returns $(F, \bar{F})$ (where $\bar{F} = M_V(F)$), then for any $S \in \mathsf{Sol}(G)$ such that $v \notin S$, $F \subseteq S$ and $S \cap \bar{F} = \emptyset$; moreover, if we define $H' = H \setminus (F \cup \bar{F})$ and $C' = C \setminus (F \cup \bar{F})$, then $G' = G[H' \cup C']$ is a crowned graph with $|C'| = |H'|$, and $S \cap (C' \cup H') \in \mathsf{Sol}(G')$.*
3. *For any $S' \in \mathsf{Sol}(G')$, $F \cup S'$ is a solution of G that avoids v.*

Proof. Suppose that there is some $S \in \mathsf{Sol}(G)$ where $v \notin S$. Let us prove by induction that for any $i \geq 1$, if there is a failure when computing F_i or $\bar{F}_i$, then S cannot exist, and otherwise $F_i \subseteq S$ and $S \cap \bar{F}_i = \emptyset$.

Assume there was no failure, and consider the step where we try to compute F_i. By induction, $S \cap \bar{F}_{i-1} = \emptyset$, so we have that $N(\bar{F}_{i-1}) \subseteq S$ and, by construction, $F_i = N(\bar{F}_{i-1})$. By Remark 6, if there is an edge $e \in M$ where $V(e) \subseteq F_i$, then no such solution exists. Let us now consider the case where we try to compute $\bar{F}_i$. Again by induction, we have $F_{i-1} \subseteq S$,

and by Remark 6 we know that no edge of M can have both endpoints in S, implying that we must have $S \cap M_V(F_i) = \emptyset$. Thus, if $G[M_V(F_i)]$ is not an independent set, then no such solution exists.

The correctness of Item 1 now directly follows from the induction. The proof of Item 2 is also immediate: if $\mathsf{propagate}_{\mathsf{all}}(G, v)$ returns $(F, \bar{F})$, then $\bar{F} = M_V(F)$ follows from the definition, and for $S \in \mathsf{Sol}(G)$ such that $v \notin S$, $F \subseteq S$ and $S \cap \bar{F} = \emptyset$ follows from the induction, and the fact that $(S \cap (C' \cup H')) \in \mathsf{Sol}(G')$ is true by Remark 6.

To prove Item 3, let $S' \in \mathsf{Sol}(G')$, and let $S = F \cup S'$. Notice that $H \cup C$ is partitioned into $F \cup \bar{F}$ and $H' \cup C'$, that F is a vertex cover of $G[F \cup \bar{F}]$ (as $\bar{F} = M_V(F)$), and that S' is a vertex cover of $G[H' \cup C']$. Now, let i_0 be the index where $\mathsf{propagate}_{\mathsf{all}}$ stops, meaning that $F_{i_0} = F_{i_0-1}$. This implies that $N(\bar{F}_{i_0}) \subseteq F_{i_0-1}$, and thus that there is no edge between $\bar{F}_{i_0} = \bar{F}$ and $H' \cup C'$, implying that S is a vertex cover of $G[H \cup C]$. As $|S| = |H|$, we get that S is a solution of G that avoids v by definition of F. ◀

With the propagation algorithm in hand, Lemma 16 can be proved by constructing a branching algorithm where we either add a vertex v of H or do not add it to the solution. The key properties we use are that: (i) there is always one solution that uses v (e.g. taking the entirety of H), and (ii) we can decide in polynomial time if it is possible to avoid v in a solution. Consequently, we recursively call our branching algorithm only if we know that there is some solution to be found in this branch.

▶ **Lemma 16.** *There is a polynomial-delay algorithm for* Enum Small Crown.

Proof. Let $G = (H \cup C, E)$ be a small crowned graph and let us define an algorithm $\mathcal{A}_s$ that enumerates all vertex covers of G of size $|H|$. If $V(G) = \emptyset$, then $\mathcal{A}_s$ returns $\emptyset$, so now let $v \in H$ be an arbitrary vertex. Let $(F_v, \bar{F}_v) = \mathsf{propagate}_{\mathsf{dir}}(G, \{v\})$, $H'_v = H \setminus F_v$, $C'_v = C \setminus \bar{F}_v$, and $G'_v = G[H'_v \cup C'_v]$. Algorithm $\mathcal{A}_s$ outputs first all solutions of the form $F_v \cup S'$, where S' are enumerated by a recursive call $\mathcal{A}_s(G'_v)$. Then, $\mathcal{A}_s$ calls $\mathsf{propagate}_{\mathsf{all}}(G, v)$. If it fails then $\mathcal{A}_s$ stops, otherwise let $(F_{\bar{v}}, \bar{F}_{\bar{v}})$ be the output of $\mathsf{propagate}_{\mathsf{all}}(G, \bar{v})$, $H'_{\bar{v}} = H \setminus F_{\bar{v}}$, $C'_{\bar{v}} = C \setminus \bar{F}_{\bar{v}}$, and $G'_{\bar{v}} = G[H'_{\bar{v}} \cup C'_{\bar{v}}]$. Algorithm $\mathcal{A}_s$ outputs then all solutions of the form $F_{\bar{v}} \cup S'$, where S' are enumerated by a recursive call $\mathcal{A}_S(G'_{\bar{v}})$. This concludes the definition of $\mathcal{A}_s$.

First, observe that the recursive calls are made on small crowned graphs, as they were obtained by removing sets $(F, \bar{F})$ from $V(G)$ where $\bar{F} = M_V(F)$. The fact that there is no repetition is immediate by induction, as in particular all solutions of $F_v \cup S'$, where $S' \in \mathsf{Sol}(G'_v)$, contain v, and all solutions of the form $F_{\bar{v}} \cup S''$, where $S'' \in \mathsf{Sol}(G'_{\bar{v}})$, do not contain v.

Let us prove by induction that all enumerated elements are in $\mathsf{Sol}(G)$. We first consider enumerated sets of the form $F_v \cup S'$, where S' are enumerated by $\mathcal{A}_s(G'_v)$. By induction, we get that $S' \in \mathsf{Sol}(G'_v)$, and by Item 3 of Lemma 11, we get that $F_v \cup S' \in \mathsf{Sol}(G)$. Now, for enumerated sets of the form $F_{\bar{v}} \cup S''$, where S'' are enumerated by $\mathcal{A}_s(G'_{\bar{v}})$, the induction hypothesis implies that $S' \in \mathsf{Sol}(G'_{\bar{v}})$, and again by Item 3 of Lemma 15, we have that $F_{\bar{v}} \cup S'' \in \mathsf{Sol}(G)$.

Let us now prove by induction that all elements of $\mathsf{Sol}(G)$ are enumerated. Let $S \in \mathsf{Sol}(G)$. If $v \in S$, then by Lemma 11, $S = F_v \cup S'$, where $S' \in \mathsf{Sol}(G'_v)$, and by induction S' will be enumerated by $\mathcal{A}_s(G'_v)$. If $v \notin S$, then by Lemma 15, as $\mathsf{propagate}_{\mathsf{all}}(G, v)$ cannot fail (as S exists), $S = F_{\bar{v}} \cup S'$, where $S' \in \mathsf{Sol}(H'_{\bar{v}}, C'_{\bar{v}})$, and by induction S' will be enumerated by $\mathcal{A}_s(H'_{\bar{v}}, C'_{\bar{v}})$.

Finally, there is at most polynomial time between any two solutions, as both $\mathsf{propagate}_{\mathsf{dir}}$ and $\mathsf{propagate}_{\mathsf{all}}$ run in polynomial time, and because for any recursive call made on an input $G' \in \{G'_v, G'_{\bar{v}}\}$, we know according to Remark 6 that $\mathsf{Sol}(G') \neq \emptyset$. Moreover, the branching

tree has height bounded by $|V(G)|$ and every node either: is a leaf, and so falls in the base case where $V(G) = \emptyset$; has exactly one child, corresponding to $\mathcal{A}_s(G'_v)$, for which a solution always exists; or has two children where $\mathcal{A}_s(G'_v)$ has the same guarantee as in the previous case, and $\mathcal{A}_s(G'_{\bar{v}})$ is called if and only if we know that $\mathsf{Sol}(G'_{\bar{v}}) \neq \emptyset$ by Lemma 15. Consequently, every leaf of the branching tree corresponds to a solution of G and the time taken between visiting two of them is bounded by a polynomial in $|V(G)| + |E(G)|$, guaranteeing the desired polynomial delay. ◀

The following theorem is now a direct consequence of Lemmas 9, 12, and 16.

▶ **Theorem 1.** Enum Vertex Cover *admits a PDE kernel with at most* $2k$ *vertices when parameterized by the maximum desired size of a solution* k*.*

4 Feedback Vertex Set

Throughout this section, we take (G, k) to be our input instance to Enum Feedback Vertex Set, let $X \subseteq V(G)$ be a 2-approximation using the algorithm given in [2], $\mathsf{fvs}(G)$ denote the size of a minimum feedback vertex set, and $\Delta(G)$ its maximum degree. Thomassé's [39] strategy can be split into four phases: (i) apply reduction rules to bound the maximum and minimum degrees of the instance; (ii) prove that $|V(G)| \in \mathcal{O}(\Delta(G) \cdot \mathsf{fvs}(G))$ in graphs of minimum degree three; (iii) identify if there is a vertex v in X that belong to too many vertex-disjoint cycles in $(G \setminus X) \cup \{v\}$; and (iv) once no such v exists, pick a high-degree vertex u, compute a 2-expansion (i.e. a crown decomposition where each head vertex is matched to two unique crown vertices) in an auxiliary graph, and remove all edges from u to the vertices identified by the expansion. As exhaustively applying (iv) implies a bound on the maximum degree, (i) and (ii) will yield the desired kernel. We follow a similar kernelization strategy, but use a substantially different set of rules beyond the basic ones; our presentation is closer to that of Fomin et al. [28, Chapter 5]. We present a generalization of point (i) in Lemma 20: $|V(G)| \in \mathcal{O}(\Delta(G) \cdot \mathsf{fvs}(G))$ holds even in some of graphs of minimum degree two.

Due to space limitations, all proofs of safeness and associated lifting algorithms of the reductions rules presented in this section have been omitted, and can be found in the full version available online.

We begin by bounding the minimum degree of G; like other works dealing with multigraphs, we define $\deg_G(v)$ as the number of edges incident to a vertex v in G, and omit the subscript when G is clear from the context. We say that $uv \in E(G)$ is a *double-edge* if the multiplicity of uv in $E(G)$, denoted by $\mathsf{mul}(uv)$, is two. Note that a double-edge is equivalent to a cycle of length two in G or, equivalently, that at least one of its endpoints must be present in *every* solution of the instance. Our first rule is only used to simplify the graph.

▶ **Reduction Rule FVS.1** ($\star$)**.** *Perform the following modifications:*

- **i.** *If there is an edge* uv *of* G *of multiplicity at least three, set its multiplicity to two.*
- **ii.** *If there is some* $v \in V(G)$ *of degree at most one, remove* v*.*
- **iii.** *If* $u, v \in V(G)$ *have* $k + 2$ *common neighbors, add edges* uv *until* $\mathsf{mul}(uv) = 2$*.*
- **iv.** *If there is some vertex* v *incident to a loop or to* $k + 1$ *distinct double-edges, remove* v *and set* $k \leftarrow k - 1$*.*

Let us briefly discuss double-edges and their implications to a Feedback Vertex Set instance. Typically, double-edges can be divided into two groups: *real* edges, where we detect that one of the endpoints in fact belongs to every solution, and *virtual* edges, that are used to simplify the instance by showing that at least one solution (if any exists) contains one

of the endpoints. The key reduction rule of Thomassé [39], which yields the required linear degree bound, makes heavy usage of virtual edges; unfortunately, we do not know how to manage the (potentially exponentially) many solutions that are forbidden in the reduced instance by the introduction of these virtual edges, or an alternative rule to linearly bound the maximum degree. More generally, virtual edges seem to make the lifting process much more complicated. As such we make a special effort to avoid them. In particular, this implies that the naive reduction rule that replaces a degree-two vertex in a triangle with a parallel edge between its neighbors is out of the question. We remark that, while this rule does have an associated lifting procedure, we hope that maintaining only real edges can be leveraged in the future to develop a smaller PDE kernel for Enum Feedback Vertex Set.

In the following rules, we always assume that we are also given an arbitrary but fixed total ordering σ of the vertices of G. This is needed to avoid repeatedly outputting a solution during lifting.

▶ **Reduction Rule FVS.2** ($\star$). *If three vertices $a, x, b \in V(G)$ form an induced path with $N(x) = \{a, b\}$ and $\sigma(a) < \sigma(b)$, then contract x into a.*

We note that the following rule does introduce a virtual double-edge, but we remark that, if applied, it will inevitably trigger Rule FVS.4, and so we do not violate our own constraints.

▶ **Reduction Rule FVS.3** ($\star$). *If $u, x, y \in V(G)$ form a triangle with $N[x] = N[y] = \{u, x, y\}$, and $\sigma(x) < \sigma(y)$, then contract y into x, i.e., make $\mathsf{mul}(ux) = 2$.*

▶ **Reduction Rule FVS.4** ($\star$). *If $u \in V(G)$ shares double-edges with $t \geq 1$ different vertices $\varepsilon_u = \{v_1, \ldots, v_t\}$ with $\deg(v_i) = 2$ for every $i \in [t]$, remove u, every v_i, and set $k \leftarrow k - 1$.*

▶ **Reduction Rule FVS.5** ($\star$). *If there is a pair of vertices $a, b \in V(G)$ and a set of degree-two vertices $\varepsilon_{ab} = \{x_1, \ldots, x_t\}$ with $N(x_i) = \{a, b\}$ for every $i \in [t]$ and $N(b) = \{a\} \cup \varepsilon_{ab}$, then remove $N[b]$ and set $k \leftarrow k - 1$.*

▶ **Lemma 17** ($\star$). *Let G be a graph and Δ its maximum degree. If none of the Rules FVS.1–FVS.5 are applicable, then $\mathsf{fvs}(G) \geq |V(G)|/(3\Delta - 3)$.*

With Lemma 17 at hand, we are tasked with bounding the maximum degree of G. Formally, a *v-flower of order t* is a family of t cycles that pairwise intersect only at vertex v. Rule FVS.6 and Lemma 18 are taken straight from [39]; we omit the proof of the latter for brevity, while we give a proof for the former as we must discuss how to lift the solutions of the reduced instance back to the input instance.

▶ **Reduction Rule FVS.6** ($\star$). *If there is a vertex $v \in V(G)$ and an v-flower of order at least $k + 1$, remove v and set $k \leftarrow k - 1$.*

▶ **Lemma 18** ([39]). *Let G a multigraph, X a feedback vertex set of G of size at most $2k$, and $x \in V(G)$. In polynomial time, it is possible to accomplish exactly one of the following:*

1. *Decide if (G, k) has $\mathsf{Sol}(G, k) = \emptyset$, i.e., the corresponding decision problem is a* no*-instance;*
2. *Find a v-flower of order $k + 1$; or*
3. *Find a set feedback vertex set H_x such that $X \setminus \{x\} \subseteq H_x \subseteq V(G) \setminus \{x\}$ of size at most $3k$.*

What is left now is to show how to leverage the third property of Lemma 18. To do so, suppose that Rule FVS.6 is not applicable, take $v \in V(G)$ with $\deg(v) > 3k(k + 1) + 5k$, let H_v be as in Lemma 18, and let $\mathcal{C}$ denote the set of connected components of $G \setminus (H_v \cup \{v\})$. Note that, as $X \subseteq H_v \cup \{v\}$, each $C_i \in \mathcal{C}$ is a tree.

▶ **Observation 19** (⋆). *There exists $\mathcal{D}_v \subseteq \mathcal{C}$ with $|\mathcal{D}_v| > 3k(k+1)$ such that, for every $D_i \in \mathcal{D}_v$: v is adjacent to one vertex of D_i with a non-double-edge, and D_i has at least one vertex adjacent to a vertex of H_v.*

At this point, we essentially have the same setup as Thomassé [39]. Instead of leveraging q-expansions, however, we take a simpler approach, which does not yield a linear bound but allows us to obtain a polynomial PDE kernel; we remark that it is still interesting to explore if these generalizations of crown decompositions can be leveraged to reduce the quadratic bound on the maximum degree that we obtain. We are first going to slightly increase the degree of v, before decreasing it, as follows.

▶ **Reduction Rule FVS.7** (⋆). *Let $\mathcal{D}_v, H_v$ be as above, and G_v be the bipartite graph with bipartition $(\{d_i \mid D_i \in \mathcal{D}_v\}, \{h_u \mid u \in H_v\})$, and $d_i h_u \in E(G_v)$ if and only if $N_G(u) \cap D_i \neq \emptyset$. For every $h_u \in V(G_v)$ of degree at least $k+2$, add a double-edge uv to G.*

We remark that the double-edges introduced by Rule FVS.7 are not virtual: there are $k+2$ cycles with pairwise intersection equal to $\{u, v\}$.

▶ **Reduction Rule FVS.8** (⋆). *Let H_v^2 be the vertices of H_v with a double-edge to v. If there exists $D_i \in \mathcal{D}_v$ with $N_G(D_i) \setminus \{v\} \subseteq H_v^2$, remove vw from G, where $w \in D_i \cap N_G(v)$.*

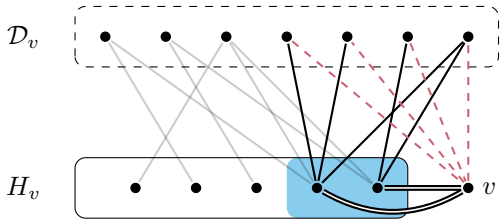


Figure 4 Example of repeated applications of Rule FVS.8; where each $D_i \in \mathcal{D}_v$ is contracted into a single vertex for ease of presentation. The two cyan-shaded vertices correspond to H_v^2. The double-edges incident to v are real double-edges, the seven light-gray edges are incident to vertices of $\mathcal{D}_v$ not affected by the rule, the four red dashed edges are the ones removed by the reduction rule, and the five solid edges are incident to vertices of $\mathcal{D}_v$ affected by the rule.

Rule FVS.8 is our key rule to reduce the maximum degree of the graph; we present an example of repeated applications of it in Figure 4. We are now ready to bound the maximum degree of G by a function of k; along with Lemma 17, this will imply a bound on the overall size of G.

▶ **Lemma 20.** *If Rules FVS.7 and FVS.8 are not applicable, then G has maximum degree at most $3k(k+1)+5k$.*

Proof. Suppose that there is some $v \in V(G)$ incident to more than $3k(k+1)+5k$ edges. By Observation 19 and Rule FVS.6, the set $\mathcal{D}_v$ of connected components of $G \setminus (H_v \cup \{v\})$ that v has a neighbor in, but no double-edges to, has size greater than $3k(k+1)$; moreover, every $D_i \in \mathcal{D}_v$ has a neighbor in H_v. Now, take some $u \in H_v$ for which $\mathsf{mul}(uv) < 2$; as Rule FVS.7 is not applicable, we know that $N_G(u)$ hits at most $k+1$ components of $\mathcal{D}_v$. As such, taking H_v^2 as in Rule FVS.8, we know that $N_G(H_v \setminus H_v^2)$ hits at most $|H_v \setminus H_v^2| \cdot (k+1)$ components of $\mathcal{D}_v$. Note that, if $H_v^2 = \emptyset$ and $|\mathcal{D}_v| > |H_v| \cdot (k+1)$, then there is some $h \in H_v$ adjacent to vertices in at least $k+2$ components of $\mathcal{D}_v$ and Rule FVS.7 is applicable, so it must be the case that $H_v^2 \neq \emptyset$. As such, since at most $(|H_v|-1)(k+1)$ components of $\mathcal{D}_v$ have a vertex adjacent to a vertex in $H_v \setminus H_v^2$, it follows that there is at least one component of $\mathcal{D}_v$ whose neighborhood is contained in $H_v^2 \cup \{v\}$, which is enough to trigger Rule FVS.8. ◀

▶ **Lemma 21.** *There is a polynomial-time algorithm that, given an instance* (G, k) *of* Enum Feedback Vertex Set*, either answers that* $\mathsf{Sol}(G, k) = \emptyset$ *or outputs an equivalent instance* (G', k') *where* $|V(G)| \leq 3k^3 + 8k^2$.

Proof. The algorithm simply consists of the exhaustive application of Rules FVS.1–FVS.8, which also guarantee that (G, k) and (G', k') are equivalent. By Lemma 20, we have that the maximum degree Δ of G' is at most $3k(k+1) + 5k$. By Lemma 17, we have that $\mathsf{fvs}(G') \geq |V(G')|/(3\Delta - 3)$. As such, if $k'(3k'(k'+1) + 5k') < |V(G')|$, then we know that no feedback vertex set of size k' exists in G', and so $\mathsf{Sol}(G', k') = \mathsf{Sol}(G, k) = \emptyset$. Otherwise, $|V(G')| \leq 3k'^3 + 3k'^2 + 5k'^2 = 3k'^3 + 8k'^2$ and we are done. ◀

▶ **Lemma 22** ($\star$)**.** *There is a polynomial-delay algorithm that, given* (G, k)*,* (G', k')*, and* $S' \in \mathsf{Sol}(G', k')$*, returns a non-empty set* $\mathsf{Lift}(S') \subseteq \mathsf{Sol}(G, k)$*. Moreover,* $\mathsf{Lift}(G', k) := \{\mathsf{Lift}(S') \mid S' \in \mathsf{Sol}(G', k')\}$ *is a partition of* $\mathsf{Sol}(G, k)$*.*

Finally, the proof of Theorem 2 follows immediately from Lemmas 21 and 22.

▶ **Theorem 2.** *There is a PDE kernel for* Enum Feedback Vertex Set *parameterized by the size of the solution* k *with* $\mathcal{O}(k^3)$ *vertices.*

5 Final remarks

In this paper, we have developed polynomial-sized PDE kernels for the natural parameterizations of enumeration variants of some key parameterized problems, namely Enum Vertex Cover and Enum Feedback Vertex Set. In this process, we showed how to leverage crown decompositions, an important technique initially developed for decision kernelization, to the enumeration setting. We hope that these results can further motivate the study of this new domain of parameterized complexity. Among concrete open problems, we are interested in the enumeration of minimal vertex covers and feedback vertex sets; we believe our techniques are applicable to these variants of the studied problems. A more challenging question is the existence of a quadratic kernel for Enum Feedback Vertex Set, which we do not see how to obtain using our approach; it seems like tools describing global structures of the graph, like q-expansions, are needed. Finally, we remark that lower bound theories for enumerative kernelization and enumerative parameterized complexity would be of extreme benefit to the area. Currently, lower bounds are derived almost solely via the related parameterized *decision* problems. It is unlikely, however, that this strategy is enough to (conditionally) classify parameterized enumeration problems into tractable and intractable and which admit polynomial PDE kernels and those that do not.

References

1 Faisal N. Abu-Khzam, Michael R. Fellows, Michael A. Langston, and W. Henry Suters. Crown structures for vertex cover kernelization. *Theory of Computing Systems*, 41(3):411–430, 2007. doi:10.1007/s00224-007-1328-0.

2 Vineet Bafna, Piotr Berman, and Toshihiro Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM Journal on Discrete Mathematics*, 12(3):289–297, 1999. doi:10.1137/S0895480196305124.

3 Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. On acyclic conjunctive queries and constant delay enumeration. In Jacques Duparc and Thomas A. Henzinger, editors, *Computer Science Logic*, pages 208–222, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. doi:10.1007/978-3-540-74915-8_18.

4 Max Bannach, Zacharias Heinrich, Rüdiger Reischuk, and Till Tantau. Dynamic kernels for hitting sets and set packing. *Algorithmica*, 84(11):3459–3488, 2022. doi:10.1007/s00453-022-00986-0.

5 Valentin Bartier, Oscar Defrain, and Fionn Mc Inerney. Hypergraph dualization with FPT-delay parameterized by the degeneracy and dimension. In *Combinatorial Algorithms: 35th International Workshop, IWOCA 2024, Ischia, Italy, July 1–3, 2024, Proceedings*, pages 111–125, Berlin, Heidelberg, 2024. Springer-Verlag. doi:10.1007/978-3-031-63021-7_9.

6 Stéphane Bessy, Marin Bougeret, Dimitrios M. Thilikos, and Sebastian Wiederrecht. *Kernelization for Graph Packing Problems via Rainbow Matching*, pages 3654–3663. Society for Industrial and Applied Mathematics, 2023. doi:10.1137/1.9781611977554.ch139.

7 Hans L. Bodlaender, Fedor V. Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M. Thilikos. (Meta) kernelization. *J. ACM*, 63(5), November 2016. doi:10.1145/2973749.

8 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM Journal on Discrete Mathematics*, 28(1):277–305, 2014. doi:10.1137/120880240.

9 Hans L. Bodlaender and Thomas C. van Dijk. A cubic kernel for feedback vertex set and loop cutset. *Theor. Comp. Sys.*, 46(3):566–597, April 2010. doi:10.1007/S00224-009-9234-2.

10 J.A. Bondy and U.S.R Murty. *Graph Theory*. Springer Publishing Company, Incorporated, 1st edition, 2008.

11 Marin Bougeret, Bart M. P. Jansen, and Ignasi Sau. Bridge-depth characterizes which minor-closed structural parameterizations of vertex cover admit a polynomial kernel. *SIAM J. Discret. Math.*, 36(4):2737–2773, 2022. doi:10.1137/21M1400766.

12 Kevin Burrage, Vladimir Estivill-Castro, Michael Fellows, Michael Langston, Shev Mac, and Frances Rosamond. The undirected feedback vertex set problem has a poly(k) kernel. In *Proceedings of the Second International Conference on Parameterized and Exact Computation*, IWPEC'06, pages 192–202, Berlin, Heidelberg, 2006. Springer-Verlag. doi:10.1007/11847250_18.

13 Jonathan F. Buss and Judy Goldsmith. Nondeterminism within P. In Christian Choffrut and Matthias Jantzen, editors, *STACS 91*, pages 348–359, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg. doi:10.1007/BFB0020811.

14 Guilherme C. M. Gomes, Emanuel Juliano, Gabriel Martins, and Vinicius F. dos Santos. Matching (Multi)Cut: Algorithms, Complexity, and Enumeration. In Édouard Bonnet and Paweł Rzążewski, editors, *19th International Symposium on Parameterized and Exact Computation (IPEC 2024)*, volume 321 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:15, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.IPEC.2024.25.

15 Miroslav Chlebík and Janka Chlebíková. Crown reductions for the minimum weighted vertex cover problem. *Discrete Applied Mathematics*, 156(3):292–312, 2008. Combinatorial Optimization 2004. doi:10.1016/j.dam.2007.03.026.

16 Nadia Creignou, Markus Kröll, Reinhard Pichler, Sebastian Skritek, and Heribert Vollmer. A complexity theory for hard enumeration problems. *Discrete Applied Mathematics*, 268:191–209, 2019. doi:10.1016/j.dam.2019.02.025.

17 Nadia Creignou, Arne Meier, Julian-Steffen Müller, Johannes Schmidt, and Heribert Vollmer. Paradigms for parameterized enumeration. *Theory of Computing Systems*, 60(4):737–758, May 2017. doi:10.1007/s00224-016-9702-4.

18 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.

19 Peter Damaschke. Parameterized enumeration, transversals, and imperfect phylogeny reconstruction. *Theoretical Computer Science*, 351(3):337–350, 2006. Parameterized and Exact Computation. doi:10.1016/j.tcs.2005.10.004.

20 Peter Damaschke. Fixed-parameter enumerability of cluster editing and related problems. *Theory of Computing Systems*, 46(2):261–283, 2010. doi:10.1007/s00224-008-9130-1.

21 Holger Dell and Dániel Marx. Kernelization of packing problems. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pages 68–81, USA, 2012. Society for Industrial and Applied Mathematics. doi:10.1137/1.9781611973099.6.

22 Rodney G Downey and Michael R Fellows. *Fundamentals of parameterized complexity*, volume 4. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

23 Arnaud Durand, Nicole Schweikardt, and Luc Segoufin. Enumerating answers to first-order queries over databases of low degree. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '14, pages 121–131, New York, NY, USA, 2014. Association for Computing Machinery. doi:10.1145/2594538.2594539.

24 Henning Fernau. On parameterized enumeration. In Oscar H. Ibarra and Louxin Zhang, editors, *Computing and Combinatorics*, pages 564–573, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. doi:10.1007/3-540-45655-4_60.

25 Fedor V. 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 Trans. Algorithms*, 15(1), January 2019. doi:10.1145/3293466.

26 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, Geevarghese Philip, and Saket Saurabh. Hitting forbidden minors: Approximation and kernelization. *SIAM Journal on Discrete Mathematics*, 30(1):383–410, 2016. doi:10.1137/140997889.

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

28 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: Theory of Parameterized Preprocessing*. Cambridge University Press, 2019. doi:10.1017/9781107415157.

29 Valentin Garnero, Christophe Paul, Ignasi Sau, and Dimitrios M. Thilikos. Explicit linear kernels via dynamic programming. *SIAM Journal on Discrete Mathematics*, 29(4):1864–1894, 2015. doi:10.1137/140968975.

30 Petr A. Golovach, Christian Komusiewicz, Dieter Kratsch, and Van B. 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.

31 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.

32 Eun Jung Kim, Alexander Langer, Christophe Paul, Felix Reidl, Peter Rossmanith, Ignasi Sau, and Somnath Sikdar. Linear kernels and single-exponential algorithms via protrusion decompositions. *ACM Trans. Algorithms*, 12(2), December 2015. doi:10.1145/2797140.

33 Christian Komusiewicz, Diptapriyo Majumdar, and Frank Sommer. Polynomial-size enumeration kernelizations for long path enumeration, 2025. doi:10.48550/arXiv.2502.21164.

34 Mithilesh Kumar and Daniel Lokshtanov. A 2lk Kernel for l-Component Order Connectivity. In Jiong Guo and Danny Hermelin, editors, *11th International Symposium on Parameterized and Exact Computation (IPEC 2016)*, volume 63 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.IPEC.2016.20.

35 A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960. URL: http://www.jstor.org/stable/1910129.

36 Diptapriyo Majumdar. Enumeration kernels of polynomial size for cuts of bounded degree, 2024. arXiv:2308.01286.

37 Kitty Meeks. Randomised enumeration of small witnesses using a decision oracle. *Algorithmica*, 81(2):519–540, 2019. doi:10.1007/s00453-018-0404-y.

38 G. L. Nemhauser and L. E. Trotter. Vertex packings: Structural properties and algorithms. *Mathematical Programming*, 8(1):232–248, 1975. doi:10.1007/BF01580444.

39 Stéphan Thomassé. A $4k^2$ kernel for feedback vertex set. *ACM Trans. Algorithms*, 6(2), April 2010. doi:10.1145/1721837.1721848.

A Note on the Parameterised Complexity of Coverability in Vector Addition Systems

Michał Pilipczuk
Institute of Informatics, University of Warsaw, Poland

Sylvain Schmitz
Université Paris Cité, CNRS, IRIF, France

Henry Sinclair-Banks
Institute of Informatics, University of Warsaw, Poland

Abstract

We investigate the parameterised complexity of the classic coverability problem for vector addition systems (VAS): given a finite set of vectors $\boldsymbol{V} \subseteq \mathbb{Z}^d$, an initial configuration $\boldsymbol{s} \in \mathbb{N}^d$, and a target configuration $\boldsymbol{t} \in \mathbb{N}^d$, decide whether starting from $\boldsymbol{s}$, one can iteratively add vectors from $\boldsymbol{V}$ to ultimately arrive at a configuration that is larger than or equal to $\boldsymbol{t}$ on every coordinate, while not observing any negative value on any coordinate along the way. We consider two natural parameters for the problem: the dimension d and the size of $\boldsymbol{V}$, defined as the total bitsize of its encoding. We present several results charting the complexity of those two parameterisations, among which the highlight is that coverability for VAS parameterised by the dimension and with all the numbers in the input encoded in unary is complete for the class XNL under PL-reductions. We also discuss open problems in the topic, most notably the question about fixed-parameter tractability for the parameterisation by the size of $\boldsymbol{V}$.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Software and its engineering → Model checking; Theory of computation → Concurrency

Keywords and phrases vector addition system, Petri net, parameterised complexity, coverability

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.24

Related Version *Full Version*: `https://arxiv.org/abs/2511.19212`

Funding *Michał Pilipczuk*: This work 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).
Henry Sinclair-Banks: This work is a part of project INFSYS that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 950398).

Acknowledgements This research was initiated during the InfAut 2024 workshop in Warlity Wielkie, Poland. The authors thank the organisers for launching such a fruitful event. They also thank an anonymous reviewer for pointing them to a relevant discussion in Hack's PhD thesis [26].

1 Introduction

Vector Addition Systems. Vector addition systems are a well-established model with a very simple definition: a d-dimensional *vector addition system* (VAS) [31] is a finite set $\boldsymbol{V}$ of vectors in $\mathbb{Z}^d$, which defines a step relation between configurations in $\mathbb{N}^d$: $\boldsymbol{u} \to_{\boldsymbol{V}} \boldsymbol{u}+\boldsymbol{v}$ for all $\boldsymbol{u}$ in $\mathbb{N}^d$ and $\boldsymbol{v}$ in $\boldsymbol{V}$, provided that $\boldsymbol{u}+\boldsymbol{v}$ is in $\mathbb{N}^d$. One usually interprets the d coordinates of a configuration $\boldsymbol{u} \in \mathbb{N}^d$ as d *counters* that may take non-negative integer values; then $\boldsymbol{V}$ represents a set of allowed updates to the counter values, which can be applied only if none of the counters becomes negative.

© Michał Pilipczuk, Sylvain Schmitz, and Henry Sinclair-Banks;
licensed under Creative Commons License CC-BY 4.0

In spite of this apparent simplicity, vector addition systems can exhibit highly complex behaviours. Famously, their *reachability problem* is ACKERMANN-complete [35, 34, 13]: given as input a d-dimensional VAS $\boldsymbol{V}$, an initial configuration $\boldsymbol{s} \in \mathbb{N}^d$, and a target configuration $\boldsymbol{t} \in \mathbb{N}^d$, the reachability problem asks whether $\boldsymbol{t}$ is reachable from $\boldsymbol{s} \in \mathbb{N}^d$ in $\boldsymbol{V}$, i.e., whether $\boldsymbol{s} \to_{\boldsymbol{V}}^* \boldsymbol{t}$. This combination of a simple definition with rich behaviours makes vector addition systems – along with the equivalent model of Petri nets – well-suited whenever one needs to model systems managing multiple discrete resources, e.g., threads in concurrent computations, molecules in chemical reactions [3, 4], organisms in biological processes [5], etc., but also as a theoretical tool involved in establishing decidability and complexity statements for a variety of decision problems in logic, formal languages, verification, etc. [42, Section 5].

The Coverability Problem. In this note, we are interested in a decision problem with a less extreme complexity: given the same input, the *coverability problem* asks whether there exists a coordinate-wise larger or equal configuration $\boldsymbol{t}' \sqsupseteq \boldsymbol{t}$ such that $\boldsymbol{s} \to_{\boldsymbol{V}}^* \boldsymbol{t}'$. This relaxation of the reachability problem was first shown decidable by Karp and Miller [31], before being proven EXPSPACE-hard by Lipton [36] and to belong to EXPSPACE by Rackoff [40].

Like reachability, coverability in vector addition systems inter-reduces with numerous decision problems, notably in relation to the automated verification of safety properties in concurrent or distributed systems [30, 29, 22, 16, 18, 23, 2, 6], as well as reasoning on data-aware logics or systems [15, 25, 1]. These applications have motivated a thorough investigation of the problem [31, 36, 40, 9, 33, 32, 44] and the development of several tools specifically targeted at solving it [19, 8, 24].

Towards Parameterised Complexity. Rosier and Yen [41] refined Rackoff's analysis to identify the contribution of several parameters to the final complexity of the coverability problem. Among those, the main parameter driving the complexity is the *dimension*, i.e., the number of counters of the system. Indeed, Rackoff obtains his result by showing a bound on the length of the shortest covering runs from the source to the target configuration, which is in $n^{2^{\mathcal{O}(d \log d)}}$ when parameterised by the dimension d, where n is the size of the input encoded in unary. (While EXPSPACE-completeness holds with both a unary and a binary encoding, the complexity landscape changes as soon as one isolates the dimension as a parameter).

This upper bound on the length of shortest covering runs in unary-encoded VAS was recently improved to $n^{2^{\mathcal{O}(d)}}$ by Künnemann, Mazowiecki, Schütze, Sinclair-Banks, and Węgrzycki [32, Theorem 3.3], and this matches the $n^{2^{\Omega(d)}}$ lower bound in the family of systems constructed by Lipton [36]. In turn, this upper bound on the length of the shortest covering runs entails that the coverability problem with a unary encoding can be solved either in non-deterministic space $2^{\mathcal{O}(d)} \cdot \log n$ or in deterministic time $n^{2^{\mathcal{O}(d)}}$ [32, Corollary 3.4]; moreover, the latter time bound is also achieved by the classical *backward coverability algorithm* [44, Corollary 4.5]. Finally, as also shown by Künnemann et al. through a parameterised reduction from the parameterised clique problem p-CLIQUE, this time bound is optimal if one assumes the Exponential Time Hypothesis: under the ETH, no algorithm running in deterministic time $n^{o(2^d)}$ for coverability may exist [32, Theorem 4.2].

Surprisingly, in spite of this long line of results focusing on coverability in vector addition systems, and in particular when isolating the dimension as a key parameter, the question of the parameterised complexity of the problem has not been considered (see Section 2.4 for literature on other parameterisations on Petri nets, which are not directly comparable).

▶ **Problem.** p-dim-COVERABILITY(VAS)
input a d-dimensional VAS $\boldsymbol{V}$, an initial configuration $\boldsymbol{s} \in \mathbb{N}^d$, and a target configuration $\boldsymbol{t} \in \mathbb{N}^d$
parameter d
question does $\boldsymbol{s}$ cover $\boldsymbol{t}$ in $\boldsymbol{V}$, i.e., does there exist $\boldsymbol{t}' \sqsupseteq \boldsymbol{t}$ such that $\boldsymbol{s} \to_{\boldsymbol{V}}^* \boldsymbol{t}'$?

From the viewpoint of parameterised complexity, when using a unary encoding, the $n^{2^{\mathcal{O}(d)}}$ deterministic time bounds from [32, 44] immediately yield that p-dim-COVERABILITY(VAS) is in XP, while the parameterised reduction from p-CLIQUE in [32] shows W[1]-hardness (see Figure 1 for an overview of the parameterised complexity classes discussed in this note).

Fixed Systems and Parameterisation by Size. A long-standing open question on decision problems for VAS, already raised by Hack [26, page 172] and revived in recent years [28, 17, 12], is whether any meaningful statements can be made about the complexity for a *fixed* VAS. In this direction, Draghici, Haase, and Ryzhikov [17] have recently shown that there exists a fixed $\boldsymbol{V}$ such that the coverability problem (and thus also the reachability problem) for $\boldsymbol{V}$ with a binary encoding of the initial/target configurations is already PSPACE-hard. This complexity of coverability in a fixed VAS is also related to a question on the length of shortest covering runs by Czerwiński [12]. A natural way to approach these questions within the framework of parameterised complexity is to ask about the complexity of coverability when parameterised by the *size* of the encoding of the system, which we denote by $\|\boldsymbol{V}\|$.

▶ **Problem.** p-size-COVERABILITY(VAS)
input a d-dimensional VAS $\boldsymbol{V}$, an input configuration $\boldsymbol{s} \in \mathbb{N}^d$, and a target configuration $\boldsymbol{t} \in \mathbb{N}^d$
parameter $\|\boldsymbol{V}\|$
question does $\boldsymbol{s}$ cover $\boldsymbol{t}$ in $\boldsymbol{V}$, i.e., does there exist $\boldsymbol{t}' \sqsupseteq \boldsymbol{t}$ such that $\boldsymbol{s} \to_{\boldsymbol{V}}^* \boldsymbol{t}'$?

This is an a priori easier problem than the one parameterised by the dimension, as there is a straightforward parameterised reduction from this problem to the one of coverability parameterised by dimension (see Remark 2.1).

Contributions. Our main contribution in this note is to refine the bounds and pinpoint the exact parameterised complexity of p-dim-COVERABILITY(VAS): the problem is XNL-complete under PL reductions with a unary encoding (see Section 3), and para-PSPACE-complete under FPT reductions with a binary encoding (see Section 4). These results are mainly applications of the rich literature dedicated to the coverability problem in vector addition systems: the upper bounds stem from the $2^{\mathcal{O}(d)} \cdot \log n$ non-deterministic space bounds from Rackoff's and subsequent works [40, 41, 32], while para-PSPACE-hardness with a binary encoding follows from the PSPACE-hardness of coverability in a fixed VAS [17, Corollary 2]. The XNL lower bound is a new proof, which relies on a result of Wehar [46] on the intersection non-emptiness problem for finite automata. Importantly given the dearth of "natural" complete problems for these two parameterised complexity classes and in particular for XNL, these results significantly enrich the library of known complete problems, as most of the inter-reducible problems we mentioned earlier also have natural parameters that correspond to the dimension.

As a consequence of these results, we also conclude that p-size-COVERABILITY(VAS) is para-PSPACE-complete when using a binary encoding (see Section 4), and in XNL when using a unary encoding (see Section 3). Our motivation here is to restate the questions about the complexity of coverability and reachability in fixed VAS [28, 17, 12] within the framework of parameterised complexity, which we discuss more extensively in Section 5.

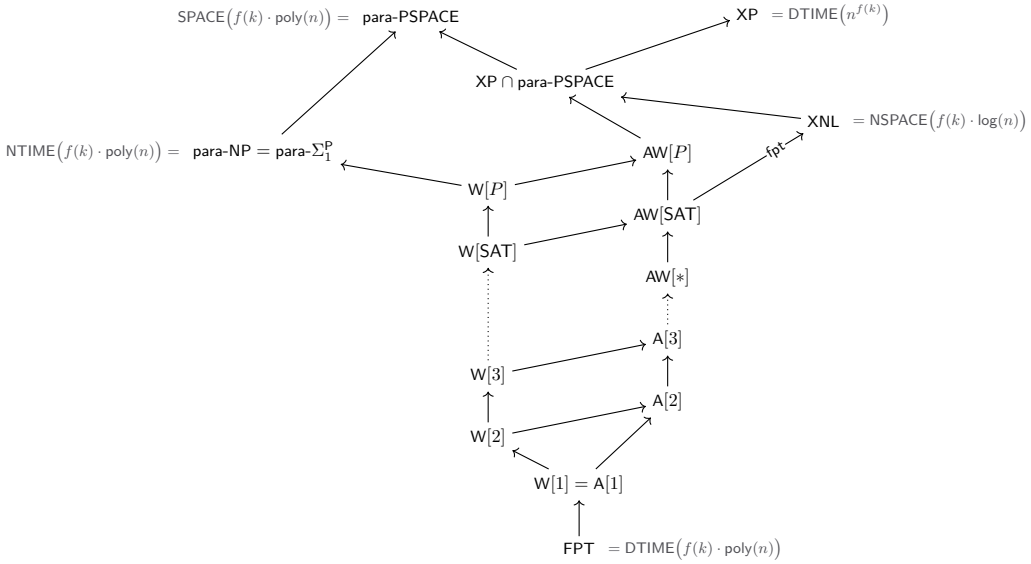


Figure 1 The parameterised complexity classes considered in this note and their known relations to the W- and A-hierarchies. Arrows C→D denote inclusions C ⊆ D; an arrow C—fpt→D denotes an inclusion C ⊆ [D]$^{\mathsf{fpt}}$ into the closure of D under FPT reductions.

2 Preliminaries

2.1 Parameterised Problems, Classes, and Reductions

Let Σ be a finite alphabet. While a decision problem is just a language $L \subseteq \Sigma^*$, a *parameterised problem* is a set $P \subseteq \Sigma^* \times \mathbb{N}$. If the pair $(x,k) \in \Sigma^* \times \mathbb{N}$ is an instance of a parameterised problem, we refer to x as the *input* to the problem and we refer to k as the *parameter*. We use the shorthand $n \stackrel{\text{def}}{=} |x|$ for the length of x, whenever this does not create confusion.

The framework of parameterised complexity has led to rich hierarchies of complexity classes, notably refining the distinction between P and NP in the presence of parameters through the class FPT of fixed-parameter tractable problems, along with classes of intractable parameterised problems: the W- and A-hierarchies, and the complexity classes para-NP and XP; see Figure 1 for a depiction of these classes. A broad introduction to parameterised complexity theory can be found in the book of Flum and Grohe [21], see also the survey of de Haan and Szeider [14] for a comprehensive overview of parameterised classes of high complexity. We provide below a basic recollection of material that will be relevant to us, based on these two sources.

Parameterised Reductions. An *FPT reduction* (respectively a *PL reduction*, which stands for *parameterised logspace)* from the parameterised problem $P_1 \subseteq \Sigma_1^* \times \mathbb{N}$ to the parameterised problem $P_2 \subseteq \Sigma_2^* \times \mathbb{N}$ is a mapping $R\colon \Sigma_1^* \times \mathbb{N} \to \Sigma_2^* \times \mathbb{N}$ such that

(1) for all $(x,k) \in \Sigma_1^* \times \mathbb{N}$, we have $(x,k) \in P_1$ if and only if $R(x,k) \in P_2$;

(2) there is a computable function $f\colon \mathbb{N} \to \mathbb{N}$ and a constant $c \in \mathbb{N}$ such that R is computable in time $f(k) \cdot n^c$ (respectively computable in space $f(k) + c \cdot \log(n)$); and

(3) there is a computable function $g\colon \mathbb{N} \to \mathbb{N}$ such that for all $(x,k) \in \Sigma_1^* \times \mathbb{N}$, say with $R(x,k) = (x',k')$, we have $k' \leqslant g(k)$.

Parameterised Space Complexity Classes. Although these classes are perhaps less well-known, the parameterised paradigm can be also applied to space complexity.

Regarding nondeterministic logarithmic space complexity, a parameterised problem $P \subseteq \Sigma \times \mathbb{N}$ is in (uniform) XNL [11, Proposition 18] if there is a computable function $f\colon \mathbb{N} \to \mathbb{N}$ and a non-deterministic algorithm that, given a pair $(x,k) \in \Sigma^* \times \mathbb{N}$, decides whether $(x,k) \in P$ in space at most $f(k) \cdot \log(n)$. In particular, AW[SAT] is included in the closure of XNL under FPT reductions by [11, Proposition 23]. When working with parameterised complexity classes like FPT, XP, para-PSPACE, etc., FPT reductions suffice, but for XNL, one needs to work with PL reductions. When stating completeness results, we always specify what type of reductions we have in mind.

Regarding polynomial space complexity, a parameterised problem $P \subseteq \Sigma^* \times \mathbb{N}$ is in para-PSPACE if there is a computable function $f\colon \mathbb{N} \to \Sigma^*$ and a problem $L \subseteq \Sigma^*$ such that $L \in$ PSPACE and for all instances $(x,k) \in \Sigma^* \times \mathbb{N}$, we have $(x,k) \in P$ if and only if $(x, f(k)) \in L$. Thus, para-PSPACE consists of all problems that can be decided using at most polynomial space after some precomputation that only involves the parameter. One can also view para-PSPACE as the space complexity-concerned analogue of FPT. In fact, it is easy to prove (see [21, Exercise 8.40]) that a parameterised problem P is in para-PSPACE if and only if it can be solved in FPT space; that is, in space bounded by $f(k) \cdot n^c$ for some computable function $f\colon \mathbb{N} \to \mathbb{N}$ and $c \in \mathbb{N}$. In order to establish para-PSPACE-hardness under FPT reductions, it suffices to prove that there exists some fixed value of the parameter k for which the problem is PSPACE-hard under polynomial-time reductions (see [21, Corollary 2.16]).

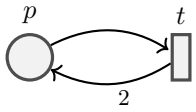
2.2 Vector Addition Systems and Related Models

As explained in the introduction, vector addition systems are equivalent to a number of models; in particular, *vector addition systems with states* are more convenient in order to prove complexity lower bounds, while *Petri nets* are better suited for modelling concurrent systems. This section succinctly introduces vector addition systems with states and Petri nets and here we argue that the usual reduction between their versions of the coverability problem holds also in the parameterised setting.

Basic Notation. We use bold font for vectors. We index the i-th component of a vector $\boldsymbol{v}$ with square brackets by writing $\boldsymbol{v}[i]$. Given two vectors $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{Z}^d$, we write $\boldsymbol{u} \sqsubseteq \boldsymbol{v}$ if $\boldsymbol{u}[i] \leqslant \boldsymbol{v}[i]$ for all $i \in \{1, \ldots, d\}$. Given a vector $\boldsymbol{v} \in \mathbb{Z}^d$, we define $\|\boldsymbol{v}\|_1 \stackrel{\text{def}}{=} |\boldsymbol{v}[1]| + \ldots + |\boldsymbol{v}[d]|$ and $\|\boldsymbol{v}\|_\infty \stackrel{\text{def}}{=} \max\{|\boldsymbol{v}[1]|, \ldots, |\boldsymbol{v}[d]|\}$. When working with unary encodings, we define the size of a vector $\boldsymbol{v} \in \mathbb{Z}^d$ as $\mathit{size}(\boldsymbol{v}) \stackrel{\text{def}}{=} \|\boldsymbol{v}\|_1$, and when using a binary encoding, as $\mathit{bitsize}(\boldsymbol{v}) \stackrel{\text{def}}{=} d \cdot (\log_2(\|\boldsymbol{v}\|_\infty + 1))$. We write $\|\boldsymbol{v}\|$ when the encoding is implicit. Then the *size* of a vector addition system $\boldsymbol{V}$ in either encoding is $\|\boldsymbol{V}\| \stackrel{\text{def}}{=} \sum_{\boldsymbol{v} \in \boldsymbol{V}} \|\boldsymbol{v}\|$, thus defining $\mathit{size}(\boldsymbol{V})$ when encoded in unary and $\mathit{bitsize}(\boldsymbol{V})$ when encoded in binary. Note that in both cases, we have $\|\boldsymbol{V}\| \geqslant d$ in a non-trivial VAS $\boldsymbol{V}$ of dimension d.

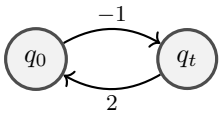
▶ Remark 2.1. For an input VAS $\boldsymbol{V}$ of some dimension d, an initial configuration $\boldsymbol{s}$, and a target configuration $\boldsymbol{t}$, the map from $(\langle \boldsymbol{V}, \boldsymbol{s}, \boldsymbol{t} \rangle, \|\boldsymbol{V}\|)$ to $(\langle \boldsymbol{V}, \boldsymbol{s}, \boldsymbol{t} \rangle, d)$ is a PL reduction (and thus also an FPT reduction) from the p-size-COVERABILITY(VAS) problem to the p-dim-COVERABILITY(VAS) problem in both a unary and a binary encoding.

Vector Addition Systems with States. A d-dimensional *vector addition system with states* (d-VASS) [27] is a pair $\mathcal{V} = (Q, T)$ consisting of a finite set of states Q and a finite set of transitions $T \subseteq Q \times \mathbb{Z}^d \times Q$. To discuss the complexity of decision problems for VASS, we define the size of a VASS $\mathcal{V} = (Q, T)$ as $\|\mathcal{V}\| \stackrel{\text{def}}{=} |Q| + \sum_{(p, \boldsymbol{x}, q) \in T} \|\boldsymbol{x}\|$.



$\{\!|p|\!\} \xrightarrow{t} \{\!|p,p|\!\}$

(a) A Petri net with a single place and a run.



$q_0(1) \to q_t(0) \to q_0(2)$

(b) A 1-VASS simulating the Petri net of Figure 2a.

$$\boldsymbol{V} \stackrel{\text{def}}{=} \left\{\boldsymbol{v}_0, \boldsymbol{v}'_0, \boldsymbol{v}_{(0,-1,t)}, \boldsymbol{v}_t, \boldsymbol{v}'_t, \boldsymbol{v}_{(t,2,0)}\right\}$$

where

$$\begin{aligned}
\boldsymbol{v}_0 &\stackrel{\text{def}}{=} (0,-1,-4,3)\\
\boldsymbol{v}'_0 &\stackrel{\text{def}}{=} (0,6,-2,-2)\\
\boldsymbol{v}_{(0,-1,t)} &\stackrel{\text{def}}{=} (-1,-4,3,-1)\\
\boldsymbol{v}_t &\stackrel{\text{def}}{=} (0,-2,-2,6)\\
\boldsymbol{v}'_t &\stackrel{\text{def}}{=} (0,3,-1,-4)\\
\boldsymbol{v}_{(t,2,0)} &\stackrel{\text{def}}{=} (2,-2,6,-2)
\end{aligned}$$

$$(1,1,6,0) \xrightarrow{\boldsymbol{v}_0\boldsymbol{v}'_0\boldsymbol{v}_{(0,-1,t)}} (0,2,3,0) \xrightarrow{\boldsymbol{v}_t\boldsymbol{v}'_t\boldsymbol{v}_{(t,2,0)}} (2,1,6,0)$$

(c) A 4-VAS simulating the VASS of Figure 2b.

Figure 2 A Petri net, its representation as a VASS, and its representation as a VAS, along with a run in all three systems.

Each index $i \in \{1,\dots,d\}$ can be seen as a counter with a valuation in $\mathbb{N}$: a *configuration* of a d-VASS is a pair $(q,\boldsymbol{v}) \in Q \times \mathbb{N}^d$ consisting of the current state q and the current counter values $\boldsymbol{v}$; we use the notation $q(\boldsymbol{v})$ for configurations. Given two configurations $p(\boldsymbol{u})$ and $q(\boldsymbol{v})$, there is a step $p(\boldsymbol{u}) \to q(\boldsymbol{v})$ if there exists a transition $t = (p,\boldsymbol{x},q) \in T$ such that $\boldsymbol{u}+\boldsymbol{x} = \boldsymbol{v}$; we may refer to $\boldsymbol{x}$ as the *update* of the transition $t = (p,\boldsymbol{x},q)$, and will also write $p(\boldsymbol{u}) \xrightarrow{t} q(\boldsymbol{v})$ to emphasise that transition t was taken from $p(\boldsymbol{u})$ to $q(\boldsymbol{v})$.

A *path* in a VASS is a sequence of transitions $((p_1,\boldsymbol{x_1},q_1),\dots,(p_m,\boldsymbol{x_m},q_m))$ such that $p_{i+1} = q_i$ for every $i \in \{1,\dots,m-1\}$; then its *length* $|\pi| \stackrel{\text{def}}{=} m$ is its number of transitions. A *run* in a VASS is a sequence of configurations $(q_0(\boldsymbol{v_0}),\dots,q_m(\boldsymbol{v_m}))$ such that, for every $i \in \{1,\dots,m\}$, $q_{i-1}(\boldsymbol{v_{i-1}}) \to q_i(\boldsymbol{v_i})$; in other words, $(q_{i-1},\boldsymbol{v_i}-\boldsymbol{v_{i-1}},q_i) \in T$. Let $\pi = (t_1,\dots,t_m)$ be a path and let $(q_0(\boldsymbol{v_0}),\dots,q_m(\boldsymbol{v_m}))$ be a run such that, for every $i \in \{1,\dots,m\}$, $q_{i-1}(\boldsymbol{v_{i-1}}) \xrightarrow{t_i} q_i(\boldsymbol{v_i})$. Then we write $q_0(\boldsymbol{v_0}) \xrightarrow{\pi} q_m(\boldsymbol{v_m})$ to denote that the run is performed along the path π, and $q_0(\boldsymbol{v_0}) \xrightarrow{*} q_m(\boldsymbol{v_m})$ if the precise path is not relevant.

The *coverability problem* for VASS takes as input a VASS $\mathcal{V}$, an initial configuration $p(\boldsymbol{u})$, and a target configuration $q(\boldsymbol{v})$, and asks whether there exists $\boldsymbol{v'} \sqsupseteq \boldsymbol{v}$ such that there is a run $p(\boldsymbol{u}) \xrightarrow{*} q(\boldsymbol{v'})$. We denote the parameterisations of this decision problem by p-dim-COVERABILITY(VASS) and p-size-COVERABILITY(VASS).

A VAS can therefore be seen as a VASS with a single state, which is dropped entirely from the definition. Conversely, by a well-known result from the seventies by Hopcroft and Pansiot, one can simulate the states of a VASS at the cost of three extra dimensions in a VAS [27, Lemma 2.1]. For clarity, the obtained VAS has an equivalent reachability relation between configurations; a configuration $q(\boldsymbol{x})$ in the original VASS corresponds to configurations of the form $(\boldsymbol{x},a_q,b_q,c_q)$ in the VAS, where a_q, b_q, and c_q are values bounded by $(|Q|+1)^2$ that represent the state q, and such that $(a_p,b_p,c_p) \not\sqsubseteq (a_q,b_q,c_q)$ for $p \neq q$ throughout the computation. Each transition of the VASS is simulated by a sequence of three transitions of the VAS, the first two checking the current state and the last one performing the update. See Figure 2 for an illustration of this construction.

▶ **Fact 2.2** (c.f. [27, Lemma 2.1]). *The parameterised problems* p-dim-COVERABILITY(VAS) *and* p-dim-COVERABILITY(VASS) *are equivalent up to PL reductions, and so are the parameterised problems* p-size-COVERABILITY(VAS) *and* p-size-COVERABILITY(VASS)*, all this both with a unary and with a binary encoding.*

Petri nets. A *Petri net* [38] is a tuple $\mathcal{N} = (P, T, W)$ where P is a finite set of places, T is a finite set of transitions, and $W\colon (P\times T)\cup(T\times P) \to \mathbb{N}$ is a (weighted) flow function. It defines a transition system with configurations in $\mathbb{N}^P$, i.e. multisets of places (aka *markings*) that can equivalently be seen as vectors in $\mathbb{N}^{|P|}$, and steps $\boldsymbol{m} \xrightarrow{t}_{\mathcal{N}} \boldsymbol{m}'$ whenever $\boldsymbol{m}(p) \geqslant W(p,t)$ and $\boldsymbol{m}'(p) = \boldsymbol{m}(p) - W(p,t) + W(t,p)$ for all p in P. We write $\boldsymbol{m} \to^*_{\mathcal{N}} \boldsymbol{m}'$ if the marking $\boldsymbol{m}'$ is reachable from the marking $\boldsymbol{m}$ by a finite sequence of such steps. The *dimension* of a Petri net is $|P|$, its number of places; its *size* is $\|\mathcal{N}\| \stackrel{\text{def}}{=} |P|+|T|+\sum_{p\in P,t\in T}(\|W(p,t)\|+\|W(t,p)\|)$. The *coverability problem* for Petri nets takes as input a Petri net $\mathcal{N}$, an initial marking $\boldsymbol{m} \in \mathbb{N}^P$, and a target marking $\boldsymbol{m}'$, and asks whether there exists $\boldsymbol{m}'' \sqsupseteq \boldsymbol{m}'$ such that $\boldsymbol{m} \to^*_{\mathcal{N}} \boldsymbol{m}''$. We denote the parameterisations of this decision problem by p-dim-COVERABILITY(PN) and p-size-COVERABILITY(PN).

See Figure 2a on Page 6 for a depiction of a Petri net $(\{p\}, \{t\}, W)$ with $W(p,t) = 1$ and $W(t,p) = 2$ as a directed bipartite graph, with places represented by circles, transitions by rectangles, and arcs from places to transitions representing the input flow $W(p,t)$ and arcs from transitions to places representing the output flows $W(t,p)$; the absence of an arc denotes a flow of 0, and an arc without weight denotes a flow of 1.

As is well-known, a Petri net (P, T, W) can be encoded as an equivalent $|P|$-dimensional VASS with $|T| + 1$ states (see Figure 2b on Page 6), and conversely a d-dimensional VAS $\boldsymbol{V}$ can be encoded as an equivalent Petri net with d places and one transition per vector $\boldsymbol{v} \in \boldsymbol{V}$, with input flow the absolute values of the negative values in $\boldsymbol{v}$ and output flow the positive values in $\boldsymbol{v}$. Thus we have the following equivalence in terms of parameterised problems.

▶ **Fact 2.3.** *The problems* p-dim-COVERABILITY(VAS) *and* p-dim-COVERABILITY(PN) *are equivalent up to PL reductions, and so are the problems* p-size-COVERABILITY(VAS) *and* p-size-COVERABILITY(PN)*, all this both with a unary and with a binary encoding.*

2.3 The Complexity of Coverability

Upper Bounds. Regarding the complexity upper bounds, consider an input of the coverability problem for a d-dimensional VAS $\boldsymbol{V}$ with initial configuration $\boldsymbol{s}$ and target configuration $\boldsymbol{t}$, encoded in unary with size $n = \mathit{size}(\boldsymbol{V})+\|\boldsymbol{s}\|_1+\|\boldsymbol{t}\|_1$. The approach pioneered by Rackoff [40] to establish the EXPSPACE upper bound is the following, and will also be pertinent for the discussion in Section 5. A finite sequence of steps $\boldsymbol{s} \to^*_{\boldsymbol{V}} \boldsymbol{t}'$ for some $\boldsymbol{t}' \sqsupseteq \boldsymbol{t}$ is a *covering run* or *coverability witness*; its length is its number of steps. What Rackoff showed is that, if there is a covering run, then there exists one of length bounded by $n^{2^{\mathcal{O}(d \log d)}}$. The best known (and actually optimal) upper bound for this length is the following.

▶ **Fact 2.4** ([32, Theorem 3.3]). *Shortest covering witnesses for an instance of coverability encoded in unary have length bounded by* $n^{2^{\mathcal{O}(d)}}$.

As a direct corollary, using algorithms that exhaustively explore the space of bounded configurations, one can determine the complexity of coverability in VAS.

▶ **Corollary 2.5** (cf. [32, Corollary 3.4]). *There exists both a deterministic* $n^{2^{\mathcal{O}(d)}}$*-time algorithm and a non-deterministic* $2^{\mathcal{O}(d)} \cdot \log(n)$*-space algorithm for coverability, where* n *is the size of the input encoded in unary.*

In the multi-parameter analysis of the complexity, one usually focuses on the deterministic algorithm, which yields that coverability can be solved in pseudo-polynomial deterministic time when the dimension is fixed. In the parameterised setting, this also yields that p-dim-COVERABILITY(VAS) with a unary encoding is in XP.

In this paper however, we are more interested in the other, non-deterministic algorithm, which we repeat from [32, Corollary 3.4] for the sake of completeness. The algorithm starts with the initial configuration and iteratively guesses the consecutive transitions in a run while keeping track of the current configuration and the total number of transitions applied so far (all the numbers are encoded in binary). If during this iteration, it encounters a configuration covering the target configuration, then it accepts, and if the bound on the length of $n^{2^{\mathcal{O}(d)}}$ provided by Fact 2.4 is exceeded without encountering a covering configuration, it rejects. The correctness of the algorithm follows directly from Fact 2.4. As for the space complexity, observe that every configuration encountered during the run will have all the counters bounded by $n^{2^{\mathcal{O}(d)}}$, and the same can be also said about the length counter. Hence, the space needed to store all the relevant information is bounded by $\mathcal{O}(d \cdot \log n^{2^{\mathcal{O}(d)}}) \leqslant 2^{\mathcal{O}(d)} \cdot \log n$.

Lower Bounds. Regarding the lower bounds, Lipton [36] was the first to show the EXPSPACE-hardness of the reachability problem, with a proof that also applies to coverability. The idea of the construction is to define by induction over d a family of VAS $\boldsymbol{V}_d$ of dimension $\mathcal{O}(d)$ that builds up counter values bounded by 2^{2^d}, allowing to simulate zero tests on those counters. This leads to a simulation of a 3-counter machine with counter values bounded by 2^{2^d}, thereby proving EXPSPACE-hardness. By a slight modification of the construction, one also obtains the following counterpart to Fact 2.4.

▶ **Fact 2.6** (c.f. [9, Theorem 3]). *Shortest covering witnesses for an instance of coverability encoded in unary may have length as large as* $n^{2^{\Omega(d)}}$.

Lipton's construction was exploited as part of a parameterised reduction from p-CLIQUE to p-dim-COVERABILITY(VAS) [32, Theorem 4.2], showing the W[1]-hardness of the problem.

2.4 Further Related Work

Watel, Weisser, and Barth [45] investigate an optimisation variant of the coverability problem in weighted Petri nets: each transition has a non-negative weight in $\mathbb{R}_{\geqslant 0}$, and the goal is to find a covering witness where the sum of weights is minimal. The parameters they consider are $\max_{p \in P, t \in T} W(p,t)$, $\max_{p \in P, t \in T} W(t,p)$ the maximal input and output flow weights, $\|\boldsymbol{m}'\|$ the size of the target marking, and the number of steps in the sought covering witness, none of which is comparable to our parameterisations by dimension or size of the Petri net.

Praveen [39] defines a graph associated with a Petri net $\mathcal{N}$ whose vertex set is P – the set of places – and edges $\{p, p'\}$ whenever there exists a transition t with $W(p,t) > 0$ and $W(t,p') > 0$. Praveen then shows that the coverability problem parameterised by both the size of a vertex cover and by the maximal flow weight $\max\{W(p,t), W(t,p) \mid p \in P, t \in T\}$ is in para-PSPACE [39, Theorem 4.5]. While this is in essence a refinement of the parameterisation by the dimension $|P|$ of the Petri net (which bounds the size of a vertex cover), the added flow weight parameter makes this result not directly comparable.

3 Unary Encoding

This section is dedicated to proving the following theorem pinpointing the exact complexity of the coverability problem in vector addition systems, when parameterised by the dimension and using a unary encoding. This is a significant improvement over the XP upper bound and W[1]-hardness mentioned in [32].

▶ **Theorem 3.1.** p-dim-COVERABILITY(VAS) *is* XNL*-complete under PL reductions when using a unary encoding.*

Using the reduction from Remark 2.1, this shows that the same upper bound also applies to the parameterisation by size.

▶ **Corollary 3.2.** p-size-COVERABILITY(VAS) *is in* XNL *when using a unary encoding.*

The membership to XNL follows from Rackoff's upper bound, that is, Fact 2.4. The XNL lower bound, however, requires some attention. We prove it using a reduction from the intersection non-emptiness problem for deterministic finite automata, which is presented in Lemma 3.3 below. To state it, we first need to establish terminology related to automata.

Finite Automata. A *deterministic finite automaton* (DFA) $\mathcal{D} = (\Sigma, Q, T, q^{\text{init}}, F)$ over a finite alphabet Σ, consists of a finite set Q of states, a set $T \subseteq Q \times \Sigma \times Q$ of transitions, an initial state $q^{\text{init}} \in Q$, and a set $F \subseteq Q$ of *final states*, with the restriction that, for every state $p \in Q$ and letter $\sigma \in \Sigma$, there is at most one state $q \in Q$ such that $(p, \sigma, q) \in T$. For a transition $t = (p, \sigma, q)$; for a word $w = \sigma_1\sigma_2 \cdots \sigma_n$ is a sequence $(q_0, \sigma_1, q_1), (q_1, \sigma_2, q_2) \cdots (a_{n-1}, \sigma_n, q_n)$; it is *accepting* if $q_n \in F$ is final. The *language* $L(\mathcal{D}) \subseteq \Sigma^*$ recognised by $\mathcal{D}$ is the set of words for which there exists some accepting run. We shall assume that a DFA is specified by its adjacency matrix, and therefore that the *size* of a given DFA $\mathcal{D} = (\Sigma, Q, T, q^{\text{init}}, F)$ is $|\mathcal{D}| \stackrel{\text{def}}{=} |Q| \cdot |\Sigma|$.

Our interest for DFAs stems from their intersection non-emptiness problem, a well-known PSPACE-complete problem, which when parameterised by the number of DFAs is one the very few known XNL-complete problems in the literature [46, Corollary 3.5].

▶ **Problem.** p-INTERSECTION-NONEMPTINESS(DFA)
input k DFAs $\mathcal{D}_1, \ldots, \mathcal{D}_k$ over the same alphabet Σ
parameter k
question is $\bigcap_{1 \leqslant i \leqslant k} L(\mathcal{D}_i) \neq \emptyset$, i.e., does there exist a word $w \in \Sigma^*$ with an accepting run in each $\mathcal{D}_i$?

Reduction to Coverability. The lower bound in Theorem 3.1 follows from the PL reduction from p-INTERSECTION-NONEMPTINESS(DFA) to p-dim-COVERABILITY(VASS) captured by the following statement.

▶ **Lemma 3.3.** *Let $\mathcal{D}_1, \ldots, \mathcal{D}_k$ be DFAs over a common alphabet Σ. One can compute a $2k$-VASS $\mathcal{V}$, an initial configuration $p(\boldsymbol{u})$, and a target configuration $q(\boldsymbol{v})$ in space $\mathcal{O}(\log(k) + \log(|\mathcal{D}_1| + \ldots + |\mathcal{D}_k|))$ such that $\bigcap_{1 \leqslant i \leqslant k} L(\mathcal{D}_i) \neq \emptyset$ if and only if there exists $\boldsymbol{v'} \geqslant \boldsymbol{v}$ and a run from $p(\boldsymbol{u})$ to $q(\boldsymbol{v'})$ in $\mathcal{V}$.*

Proof. Suppose $\mathcal{D}_i = (\Sigma, Q_i, T_i, q_i^{\text{init}}, F_i)$ for every $1 \leqslant i \leqslant k$. Without loss of generality, we shall assume that $s = |Q_1| = \ldots = |Q_k|$. It will also be convenient for us to number the states; $Q_i = \{q_{i,1}, \ldots, q_{i,s}\}$ and we may also assume without loss of generality that $q_i^{\text{init}} = q_{i,1}$.

Overview. We construct a VASS $\mathcal{V}$ with $2k$ counters that come in k pairs: for each $i \in \{1, \ldots, k\}$, there are counters $\mathsf{x_i}$ and $\mathsf{y_i}$ used to store the current state of automaton $\mathcal{D}_i$. Precisely, that the current state of $\mathcal{D}_i$ is $q_{i,j}$, for some $1 \leqslant j \leqslant s$, is reflected by counter values $\mathsf{x_i} = j$ and $\mathsf{y_i} = s - j$. The construction of $\mathcal{V}$ ensures that, except for some intermediate configurations, the following invariant is maintained: for each $i \in \{1, \ldots, k\}$, we have $\mathsf{x_i} + \mathsf{y_i} = s$ and $\mathsf{x_i} \geqslant 1$.

With this way of encoding of a selection of states of $\mathcal{D}_1, \ldots, \mathcal{D}_k$, we emulate applying a transition of any $\mathcal{D}_i$ through a pair of transitions in $\mathcal{V}$ as follows. Suppose we would like to apply a transition of $\mathcal{D}_i$ that goes from $q_{i,a}$ to $q_{i,b}$, for some $a, b \in \{1, \ldots, s\}$: then, we may apply two transitions in $\mathcal{V}$

- first, one that subtracts a from $\mathsf{x_i}$ and $s - a$ from $\mathsf{y_i}$, and
- then, one that adds b to x_i and $s - b$ to $\mathsf{y_i}$.

Thanks to the invariant, the first transition can be fired only if we indeed have $\mathsf{x_i} = a$ and $\mathsf{y_i} = s - a$ (and the state encoded by those counters is $q_{i,a}$), and firing it brings both counters to 0. Then the second transition sets $\mathsf{x_i} = b$ and $\mathsf{y_i} = s - b$. Thus, firing the two transitions in $\mathcal{V}$ both verifies that the current state is $q_{i,a}$, and updates this state to $q_{i,b}$.

With this basic gadget understood, we may explain the overall construction of $\mathcal{V}$, depicted in Figure 3. We create two special states: the starting state p and the ending state q. The initial configuration is $p(\boldsymbol{u})$ where $\boldsymbol{u} \stackrel{\text{def}}{=} (1, s-1, \ldots, 1, s-1)$, which corresponds to setting every automaton $\mathcal{D}_i$ to its initial state. Next, for each $\sigma \in \Sigma$ we construct a *σ-section*, which is essentially a loop that goes from p back to p, and a *checking section*, which is essentially a path from p to q. Thus, any run from p to q first loops some number of times through the σ-sections, and finally proceed through the checking section to q. The intention is that the effect of performing a loop through the σ-section, for some $\sigma \in \Sigma$, is an update of the values of the counters of $\mathcal{V}$ that corresponds to performing a σ-transition in each of the automata $\mathcal{D}_1, \ldots, \mathcal{D}_k$. This is done within k consecutive parts of the σ-section, each responsible for updating the state of a different $\mathcal{D}_i$ using the gadgets described in the previous paragraph. Thus, choosing consecutive loops through the σ-sections corresponds to choosing consecutive letters of a word w over Σ, and simulating the runs of all the automata $\mathcal{D}_1, \ldots, \mathcal{D}_k$ on w. Finally, we construct the checking section using a similar method so that it can be traversed if and only if each automaton $\mathcal{D}_i$ is in a final state; this verifies that w is in the intersection of the languages of $\mathcal{D}_1, \ldots, \mathcal{D}_k$. Again, the checking section consist of k parts, each responsible for checking the state of a different $\mathcal{D}_i$. See Section 3.1 for an illustration of this construction.

Formal Construction. We start by constructing the two special states p and q. Next, for every $\sigma \in \Sigma$ we construct the σ-section. It contains one part for each $i \in \{1, \ldots, k\}$ defined as follows. Let $m_{\sigma,i}$ be the number of σ-transitions in $\mathcal{D}_i$. Then we introduce $m_{\sigma,i} + 2$ many states: one starting state, one ending state, and one intermediate state for each σ-transition in $\mathcal{D}_i$. Furthermore, for every σ-transition $t = (q_{i,a}, \sigma, q_{i,b})$ in $\mathcal{D}_i$, we construct two transitions in $\mathcal{V}$: the first transition from the starting state to the intermediate state q_t that updates the counters by subtracting a from $\mathsf{x_i}$ and $s - a$ from $\mathsf{y_i}$ (and does not update the other counters); the second transition from the intermediate state q_t to the ending state that updates the counters similarly by adding b to $\mathsf{x_i}$ and $s - b$ to $\mathsf{y_i}$ (and does not update the other counters). Finally, we connect all the parts of the σ-section into a path as follows by adding for each $i \in \{1, \ldots, k-1\}$, a counter-neutral (not changing the values of the counters) transition from the ending state of the i-th part to the starting state of the $(i+1)$-st part. The section is connected to the rest of the VASS by counter-neutral transitions from p to the starting state of the first part, and from the ending state of the last part to p.

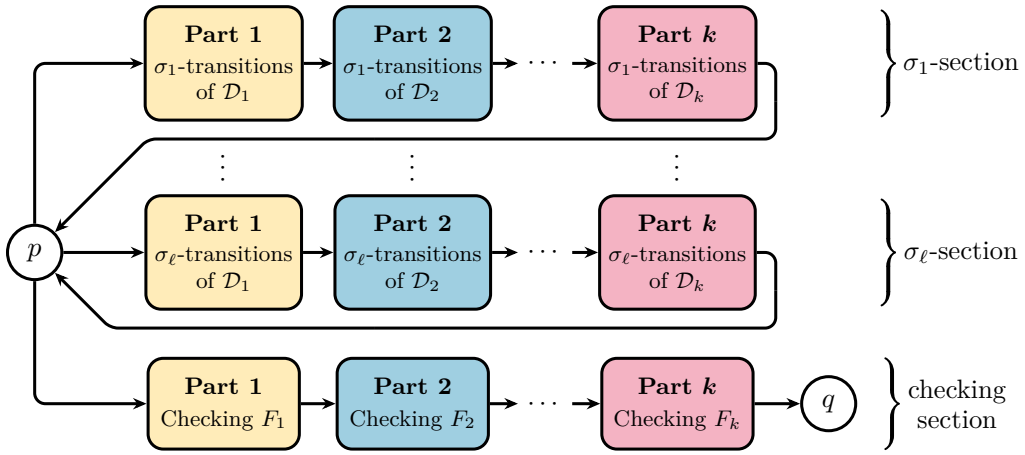


Figure 3 Overall structure of $\mathcal{V}$ in the proof of Lemma 3.3.

Next, we present the construction of the checking section, which amounts to constructing the i-th part, for each $i \in \{1, \ldots, k\}$, each with two states: one starting state and one ending state. Further, we construct $|F_i|$ many transitions: one for each final state of $\mathcal{D}_i$. Precisely, for each final state $q_{i,f}$ of $\mathcal{D}_i$ (for some $f \in \{1, \ldots, s\}$), we construct a transition in $\mathcal{V}$ from the starting state to the ending state that subtracts f from x_i and $s - f$ from y_i (and does not update the other counters). It is important to note that this is only possible if $\mathsf{x}_\mathsf{i} = f$ and $\mathsf{y}_\mathsf{i} = s - f$ (i.e., when $\mathcal{D}_i$ is currently at f). Accordingly, the only possible way to go is through the i-th part of the checking section is if $\mathcal{D}_i$ is currently in a final state. We connect again all the parts of the checking section into a path by adding counter-neutral transitions from p to the starting state of the first part, from the ending state of the last part to q, and from the ending state of the i-th part to the starting state of the $(i+1)$-st part, for each $i \in \{1, \ldots, k-1\}$.

Finally, we set the initial configuration of $\mathcal{V}$ to $p(\boldsymbol{u})$, where $\boldsymbol{u}$ sets $\mathsf{x}_\mathsf{i} = 1$ and $\mathsf{y}_\mathsf{i} = s - 1$ for every $i \in \{1, \ldots, k\}$, and the target configuration to $q(\mathbf{0})$ (so $\boldsymbol{v} = \mathbf{0}$). The following claim verifies the correctness of the reduction.

▷ Claim 3.4. The following conditions are equivalent.

- There exists a word over Σ that is accepted by all the automata $\mathcal{D}_1, \ldots, \mathcal{D}_k$.
- There is run of $\mathcal{V}$ from configuration $p(\boldsymbol{u})$ to some configuration with state q.

The proof of Claim 3.4 follows easily from the construction along the lines sketched at the beginning of the proof; we give a more formal explanation in Section A.1.

Space Complexity. Let us finally observe that the VASS $\mathcal{V}$ can be constructed in space $\mathcal{O}\big(\log(k) + \log(|\mathcal{D}_1| + \ldots + |\mathcal{D}_k|)\big)$, because the states and transitions of $\mathcal{V}$ can be computed using three pointers

- $i \in \{1, \ldots, k\}$ for the current automaton $\mathcal{D}_i$,
- $\sigma \in \Sigma$ for the current letter, and
- $q_{i,a} \in Q_i$, which together with σ determines the transition $(q_{i,a}, \sigma, q_{i,b})$ in the deterministic automaton $\mathcal{D}_i$.

This sums up to $\mathcal{O}\big(\log(k \cdot (|\Sigma| \cdot |Q_1| + \ldots + |\Sigma| \cdot |Q_k|)\big)$ space for the reduction; as furthermore $|\mathcal{D}_i| = |\Sigma| \cdot |Q_i|$ for each i, this is indeed the same as $\mathcal{O}\big(\log(k) + \log(|\mathcal{D}_1| + \ldots + |\mathcal{D}_k|)\big)$ space. ◀

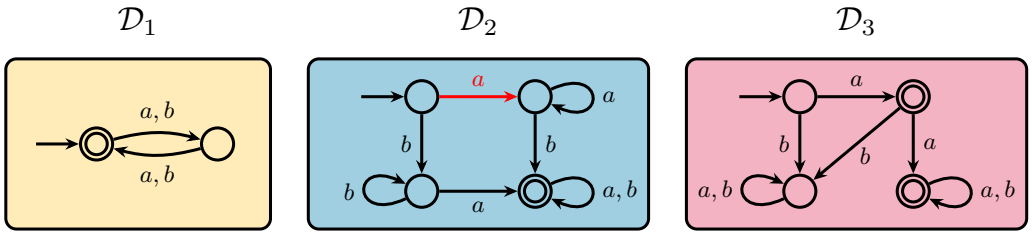


Figure 4 Three DFAs $\mathcal{D}_1$, $\mathcal{D}_2$, and $\mathcal{D}_3$.

We may now conclude this section by proving Theorem 3.1.

Proof of Theorem 3.1. Membership in XNL follows from the algorithm in nondeterministic space $2^{\mathcal{O}(d)} \cdot \log n$ from Corollary 2.5 (c.f. [32, Corollary 3.4]). The XNL-hardness follows from Fact 2.2 and the PL reduction described by Lemma 3.3 along with the XNL-hardness of p-INTERSECTION-NONEMPTINESS(DFA) proven by Wehar [46, Corollary 3.5]. ◀

3.1 Example

Let us illustrate the construction on the three DFAs over the alphabet $\{a, b\}$ of Figure 4, and construct a 6-VASS for which coverability between two specified configurations holds if and only if the intersection of the languages of the three DFAs is non-empty.

The languages of the three DFAs are the following.

- $\mathcal{D}_1$ accepts all words of even length.
- $\mathcal{D}_2$ accepts all words that contain at least one "a" and at least one "b".
- $\mathcal{D}_3$ accepts all non-empty words where the first and second letter is not "b".

The intersection of these languages is not empty; the shortest words in the intersection of their languages are "$aaab$" and "$aaba$".

The resulting 6-VASS $\mathcal{V}$ is presented in Figure 5. We set $s = 4$ to be the greatest number of states of any of the DFAs. The VASS has three sections: the a-section (the top row of the figure), the b-section (the middle row of the figure), and the checking section (the bottom row of the figure). There are three parts in each section, one for each of the three DFAs.

First, let us consider the second part in the a-section; here there is one pair of transitions for each a-transition in $\mathcal{D}_2$. For example, consider the transitions *coloured in red* in Figure 4 and Figure 5. The red pair of transitions in $\mathcal{V}$ corresponds to the red a-transition from the first state (top left) to the second state (top right) in $\mathcal{D}_2$. The two transitions in $\mathcal{V}$ have the updates $(0, 0, -1, -3, 0, 0)$ and $(0, 0, 2, 2, 0, 0)$, respectively. The first red transition in $\mathcal{V}$ checks if the current state of $\mathcal{D}_2$ is the first state ($\mathsf{x}_2 = 1$ and $\mathsf{y}_2 = s - 1 = 3$). The second red transition in $\mathcal{V}$ then updates the counters so that the current state of $\mathcal{D}_2$ is the second state ($\mathsf{x}_2 = 2$ and $\mathsf{y}_2 = s - 2 = 2$).

Second, let us consider the third part of the checking section; here there are two transitions, one for each of the two final states of $\mathcal{D}_3$. If the current counter values have $\mathsf{x}_3 = 2$ and $\mathsf{y}_3 = s - 2 = 2$ or the counter values have $\mathsf{x}_3 = 4$ and $\mathsf{y}_3 = s - 4 = 0$, then it is would be possible to pass through this part. Indeed, when the counter values of $\mathcal{V}$ contain $\mathsf{x}_3 = 2$ and $\mathsf{y}_3 = 2$, then the current state of $\mathcal{D}_3$ is the second state (top right) which is a final state. Similarly, when the counter values of $\mathcal{V}$ contain $\mathsf{x}_3 = 4$ and $\mathsf{y}_3 = 0$, then the current state of $\mathcal{D}_3$ is the fourth state (bottom right) which is also a final state.

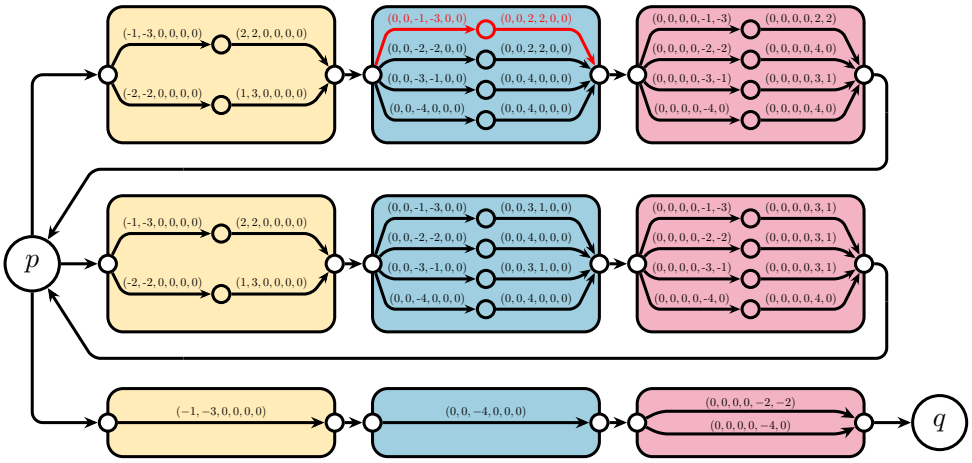


Figure 5 The 6-VASS $\mathcal{V}$ constructed from the DFAs of Figure 4.

As the intersection of the languages recognised by the DFAs is non-empty, we know by Claim 3.4 that there is a run from $p(1,3,1,3,1,3)$ to $q(0,0,0,0,0,0)$. The run corresponding to the word *aaab* follows a path of over 50 transitions; listing every intermediate configuration of the run is impractical, so we list below the main features of the run.

1. From $p(1,3,1,3,1,3)$, pass through the a-section to reach $p(2,2,2,2,2,2)$.
2. From $p(2,2,2,2,2,2)$, pass through the a-section to reach $p(1,3,2,2,4,0)$.
3. From $p(1,3,2,2,4,0)$, pass through the a-section to reach $p(2,2,2,2,4,0)$.
4. From $p(2,2,2,2,4,0)$, pass through the b-section to reach $p(1,3,4,0,4,0)$.
5. From $p(1,3,4,0,4,0)$, pass through the checking section to finally reach $q(0,0,0,0,0,0)$.

4 Binary Encoding

In this section, we show the para-PSPACE-completeness of the coverability problem parameterised either by dimension or size, when the input is encoded in binary. As in the case of a unary encoding, the upper bound follows from the $2^{O(d)} \cdot \log n$ non-deterministic space bounds from Rackoff's and subsequent works [40, 41, 32]. Regarding the lower bound, we rely on the PSPACE-hardness of coverability in a fixed VAS shown by Draghici, Haase, and Ryzhikov [17].

▶ **Theorem 4.1.** *The problems* p-dim-COVERABILITY(VAS) *and* p-size-COVERABILITY(VAS) *are* para-PSPACE*-complete under FPT reductions when using a binary encoding.*

Proof. Regarding the upper bound, recall that the coverability problem can be solved in non-deterministic space $2^{\mathcal{O}(d)} \cdot \log n$ where d is the dimension and $n = \mathit{size}(\boldsymbol{V}) + \|\boldsymbol{s}\|_1 + \|\boldsymbol{t}\|_1$ is the size of the input in unary [32, Corollary 3.4]. This in turn provides an algorithm working in non-deterministic space $2^{\mathcal{O}(d)} \cdot N$ where $N = \mathit{bitsize}(\boldsymbol{V}) + d\big(\log_2(\|\boldsymbol{s}\|_\infty + 1) + \log_2(\|\boldsymbol{t}\|_\infty + 1)\big)$ is the size of the input in binary, showing that p-dim-COVERABILITY(VAS) is in para-PSPACE. By the parameterised reduction from Remark 2.1, this also shows that p-size-COVERABILITY(VAS) is in para-PSPACE.

Regarding the lower bound, [17, Corollary 2] shows that there exists a fixed VASS $\mathcal{V}_0$ (thus of fixed size, and actually of dimension 6) such that the coverability problem for $\mathcal{V}_0$ is PSPACE-hard under the assumption that the starting and the target configuration are

encoded in binary. By Fact 2.2, there is a fixed VAS $\boldsymbol{V}_0$ (of dimension 9) with the same property, thus showing that p-size-COVERABILITY(VAS) and p-dim-COVERABILITY(VAS) are para-PSPACE-hard under FPT reductions.[1] ◀

A consequence is that, while coverability with either parameterisation is in XP when encoded in unary, if the problem with a binary encoding were to belong to XP, then P = PSPACE. Indeed, if this were the case, then coverability in the fixed VAS $\boldsymbol{V}_0$ provided by [17, Corollary 2] would be in P.

5 Discussion and Open Problems

Our results shed some light on the broader landscape of the parameterised complexity of decision problems related to vector addition systems. Let us consider the coverability problem and the reachability problem; both have meaningful parameterisations either by dimension or by size. A first set of questions is simply whether the parameterised problem is in FPT or not. A second set of questions concerns bounding the length of witness runs, and is motivated by the discussion in Section 2.3, where we saw that this approach has been instrumental in the understanding of the complexity of coverability. Given an initial configuration $\boldsymbol{s}$ and a target configuration $\boldsymbol{t}$ in a VAS $\boldsymbol{V}$, a *shortest witness* is a run $\boldsymbol{s} \to^*_{\boldsymbol{V}} \boldsymbol{t}'$ for some $\boldsymbol{t}' \sqsupseteq \boldsymbol{t}$ in the case of coverability or a run $\boldsymbol{s} \to^*_{\boldsymbol{V}} \boldsymbol{t}$ in the case of reachability, such that the number of steps in the run is minimal. For a parameter k (the dimension or the size of $\boldsymbol{V}$), we will say that shortest witnesses have *fixed-parameter length* (FPT length) if there is a computable function f and constant $c \in \mathbb{N}$ such that shortest witnesses have length at most $f(k) \cdot n^c$.

This setup leads to four questions, that can each be asked with a parameterisation by dimension or by size, and using a unary or a binary encoding.

(CP) Is the parameterised coverability problem in FPT?
(RP) Is the parameterised reachability problem in FPT?
(CL) Are shortest coverability witnesses of FPT length?
(RL) Are shortest reachability witnesses of FPT length?

Negative Cases. Quite a few of these questions have negative answers. In the case of parameterisation by dimension, we have the following:

- A positive answer to **(CP)** would entail XNL ⊆ FPT in the case of a unary encoding by Theorem 3.1, which would bring in particular a collapse of the W- and A-hierarchies down to FPT, because AW[SAT] ⊆ [XNL]$^{\text{fpt}}$ by [11, Proposition 23] (recall Figure 1). It would entail FPT = para-PSPACE in the case of a binary encoding by Theorem 4.1, which is if and only if P = PSPACE.
- **(CL)** has an unconditional negative answer regardless of the choice of encoding by Fact 2.6: Lipton's construction [36] yields a family of inputs to the coverability problem in dimension $\mathcal{O}(d)$ where the shortest witnesses have length at least n^{2^d}.
- **(RP)** and **(RL)** have a negative answer regardless of the choice of encoding since the reachability problem in dimension $\mathcal{O}(d)$ is hard for the fast-growing complexity class $\mathbf{F}_d$ [34, 13], and the shortest witnesses in their instances have length at least $F_d(n)$ where F_d is the dth fast-growing function (see [43] for a reference on fast-growing complexity).

[1] The para-PSPACE-hardness of p-dim-COVERABILITY(VAS) also follows from the fact that coverability in binary-encoded 2-VASS is PSPACE-hard [20, 7], which implies that coverability in binary-encoded 5-VAS is PSPACE-hard.

Turning our attention to parameterisation by size and a binary encoding, we have:

- A positive answer to either **(CP)** or **(RP)** would again entail P = PSPACE by Theorem 4.1.
- Finally, **(CL)** and **(RL)** have a negative answer, since with a binary encoding and a fixed VAS of constant size, one can easily ensure that shortest witnesses have length exponential in the size of the input.

Open Questions. Our questions are therefore only relevant in the case of a unary encoding and a parameterisation by size. In that setting, **(CL)** is essentially Question 3 by Czerwiński [12][2] and **(RL)** is essentially a conjecture due Hack [26, page 172] that has recently been restated by Jecker [28]. Both of these questions were originally phrased in terms of an $\mathcal{O}(n)$ upper bound in a fixed VAS; here we cast them in the terminology of parameterised complexity.

Those two questions are connected to **(CP)** and **(RP)**: an FPT length on shortest witnesses implies the existence of a nondeterministic algorithm that guesses and checks this witness. With a unary encoding, this entails that the problem is in para-NP, while with a binary encoding, this entails that the problem is in para-PSPACE. Recall that all we know at the moment are XNL and para-PSPACE upper bounds for coverability with unary and binary encoding, and no reasonable upper bounds for reachability. Thus **(CL)** and **(RL)** provide an original angle of attack on **(CP)** and **(RP)**, respectively, and a positive answer would yield

- the para-PSPACE-completeness of reachability parameterised by size with a binary encoding, and
- an indication that the complexity of the problems parameterised by size with a unary encoding is likely lower than XNL or para-NP.

In summary, we believe that **(CP)** and **(RP)** are excellent open questions in parameterised complexity that deserve further investigation, whereas **(CL)** and **(RL)** are auxiliary conjectures whose resolution might be very insightful for **(CP)** and **(RP)**.

As a further indication on the easiness of coverability and reachability parameterised by size with a unary encoding, note that in a *fixed* VAS $\boldsymbol{V}_0$, the coverability problem and the reachability problem are *sparse* languages, meaning that there exists a polynomial p such that, for all n, the number of instances x of the language of length $|x| = n$ is bounded by $p(n)$. Indeed, as $\boldsymbol{V}_0$ and thus its dimension d are fixed, there are at most $\binom{n}{2d} \leqslant n^{2d}$ possible pairs of initial and target configurations such that $\|\boldsymbol{s}\|_1 + \|\boldsymbol{t}\|_1 = n$. This tells us that if there exists a fixed $\boldsymbol{V}_0$ such that either coverability or reachability for $\boldsymbol{V}_0$ with unary encoding is NP-hard, then P = NP by Mahaney's Theorem [37, Theorem 3.1]. Similarly, NL-hardness entails L = NL [10, Corollary 3].

References

1 Parosh Aziz Abdulla, C. Aiswarya, Mohamed Faouzi Atig, Marco Montali, and Othmane Rezine. Complexity of reachability for data-aware dynamic systems. In *Proceedings of ACSD 2018*, pages 11–20. IEEE Computer Society, 2018. doi:10.1109/ACSD.2018.000-3.

2 Kalev Alpernas, Aurojit Panda, Alexander Rabinovich, Mooly Sagiv, Scott Shenker, Sharon Shoham, and Yaron Velner. Some complexity results for stateful network verification. *Formal Methods in System Design*, 54:191–231, 2019. doi:10.1007/s10703-018-00330-9.

[2] Hack observed [26, page 171] that shortest coverability witnesses were of linear length if, in addition to the VAS, the initial configuration was fixed (thanks to the coverability tree construction [31]). On a similar note, Rackoff's analysis [40] even yields a constant bound on the length of shortest coverability witnesses if we fix both the VAS and the target configuration.

3 Rutherford Aris. Prolegomena to the rational analysis of systems of chemical reactions. *Archive for Rational Mechanics and Analysis*, 19(2):81–99, 1965. doi:10.1007/BF00282276.

4 Rutherford Aris. Prolegomena to the rational analysis of systems of chemical reactions II. Some addenda. *Archive for Rational Mechanics and Analysis*, 27(5):356–364, 1968. doi:10.1007/BF00251438.

5 Paolo Baldan, Nicoletta Cocco, Andrea Marin, and Marta Simeoni. Petri nets for modelling metabolic pathways: A survey. *Natural Computing*, 9(4):955–989, 2010. doi:10.1007/S11047-010-9180-6.

6 Pascal Baumann, Rupak Majumdar, Ramanathan S. Thinniyam, and Georg Zetzsche. Context-bounded verification of thread pools. In *Proceedings of POPL 2012*, volume 6 of *Proceedings of the ACM on Programming Languages*. ACM, 2022. doi:10.1145/3498678.

7 Michael Blondin, Alain Finkel, Stefan Göller, Christoph Haase, and Pierre McKenzie. Reachability in two-dimensional vector addition systems with states is PSPACE-complete. In *Proceedings of LICS 2015*, pages 32–43. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.14.

8 Michael Blondin, Alain Finkel, Christoph Haase, and Serge Haddad. Approaching the coverability problem continuously. In *Proceedings of TACAS 2016*, volume 9636 of *Lecture Notes in Computer Science*, pages 480–496. Springer, 2016. doi:10.1007/978-3-662-49674-9_28.

9 Laura Bozzelli and Pierre Ganty. Complexity analysis of the backward coverability algorithm for VASS. In *Proceedings of RP 2011*, volume 6945 of *Lecture Notes in Computer Science*, pages 96–109. Springer, 2011. doi:10.1007/978-3-642-24288-5_10.

10 Jin-Yi Cai and D. Sivakumar. Resolution of Hartmanis' conjecture for NL-hard sparse sets. *Theoretical Computer Science*, 240(2):257–269, 2000. doi:10.1016/S0304-3975(99)00234-0.

11 Yijia Chen, Jörg Flum, and Martin Grohe. Bounded nondeterminism and alternation in parameterized complexity theory. In *Proceedings of CCC 2003*, pages 13–29. IEEE Computer Society, 2003. doi:10.1109/CCC.2003.1214407.

12 Wojciech Czerwiński. Open problems in infinite-states systems, 2024. Webpage accessed on 2024-12-07. URL: https://www.mimuw.edu.pl/~wczerwin/research.html.

13 Wojciech Czerwiński and Łukasz Orlikowski. Reachability in vector addition systems is ACKERMANN-complete. In *Proceedings of FOCS 2021*, pages 1229–1240. IEEE Computer Society, 2022. doi:10.1109/FOCS52979.2021.00120.

14 Ronald de Haan and Stefan Szeider. A compendium of parameterized problems at higher levels of the polynomial hierarchy. *Algorithms*, 12(9):188, 2019. doi:10.3390/A12090188.

15 Normann Decker, Peter Habermehl, Martin Leucker, and Daniel Thoma. Ordered navigation on multi-attributed data words. In *Proceedings of CONCUR 2014*, volume 8704 of *Lecture Notes in Computer Science*, pages 497–511. Springer, 2014. doi:10.1007/978-3-662-44584-6_34.

16 Roberto Di Cosmo, Jacopo Mauro, Stefano Zacchiroli, and Gianluigi Zavattaro. Component reconfiguration in the presence of conflicts. In *Proceedings of ICALP 2013*, volume 7966 of *Lecture Notes in Computer Science*, pages 187–198. Springer, 2013. doi:10.1007/978-3-642-39212-2_19.

17 Andrei Draghici, Christoph Haase, and Andrew Ryzhikov. Reachability in fixed VASS: expressiveness and lower bounds. In *Proceedings of FoSSaCS 2024*, volume 14575 of *Lecture Notes in Computer Science*, pages 185–205. Springer, 2024. doi:10.1007/978-3-031-57231-9_9.

18 Javier Esparza. Keeping a crowd safe: On the complexity of parameterized verification. In *Proceedings of STACS 2014*, volume 25 of *Leibniz International Proceedings in Informatics*, pages 1–10. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2014. doi:10.4230/LIPIcs.STACS.2014.1.

19 Javier Esparza, Ruslán Ledesma-Garza, Rupak Majumdar, Philipp Meyer, and Filip Niksic. An SMT-based approach to coverability analysis. In *Proceedings of CAV 2014*, volume 8559 of *Lecture Notes in Computer Science*, pages 603–619. Springer, 2014. doi:10.1007/978-3-319-08867-9_40.

20 John Fearnley and Marcin Jurdziński. Reachability in two-clock timed automata is PSPACE-complete. *Information and Computation*, 243:26–36, 2015. doi:10.1016/j.ic.2014.12.004.

21 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. Springer, 2006. doi:10.1007/3-540-29953-X.

22 Pierre Ganty and Rupak Majumdar. Algorithmic verification of asynchronous programs. *ACM Transactions on Programming Languages and Systems*, 34(1), 2012. doi:10.1145/2160910.2160915.

23 Gilles Geeraerts, Alexander Heußner, and Jean-François Raskin. On the verification of concurrent, asynchronous programs with waiting queues. *ACM Transactions on Embedded Computing Systems*, 14(3), 2015. doi:10.1145/2700072.

24 Thomas Geffroy, Jérôme Leroux, and Grégoire Sutre. Occam's Razor applied to the Petri net coverability problem. *Theoretical Computer Science*, 750:38–52, 2018. doi:10.1016/j.tcs.2018.04.014.

25 Radu Grigore and Nikos Tzevelekos. History-register automata. *Logical Methods in Computer Science*, 12(1), 2016. doi:10.2168/LMCS-12(1:7)2016.

26 Michel Hack. *Decidability questions for Petri Nets*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1976. URL: https://hdl.handle.net/1721.1/27441.

27 John Hopcroft and Jean-Jacques Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theoretical Computer Science*, 8(2):135–159, 1979. doi:10.1016/0304-3975(79)90041-0.

28 Ismaël Jecker. Open problems in automata theory: 22.1 Complexity of fixed VAS reachability, 2022. Webpage accessed on 2025-05-13. URL: https://automata.exchange/22.01-complexity-fixed-vas-reachability/.

29 Alexander Kaiser, Daniel Kroening, and Thomas Wahl. Dynamic cutoff detection in parameterized concurrent programs. In *Proceedings of CAV 2010*, volume 6174 of *Lecture Notes in Computer Science*, pages 645–659. Springer, 2010. doi:10.1007/978-3-642-14295-6_55.

30 Max Kanovich, Paul Rowe, and Andre Scedrov. Policy compliance in collaborative systems. In *Proceedings of CSF 2009*, pages 218–233. IEEE Computer Society, 2009. doi:10.1109/CSF.2009.19.

31 Richard M. Karp and Raymond E. Miller. Parallel program schemata. *Journal of Computer and System Sciences*, 3(2):147–195, 1969. doi:10.1016/S0022-0000(69)80011-5.

32 Marvin Künnemann, Filip Mazowiecki, Lia Schütze, Henry Sinclair-Banks, and Karol Węgrzycki. Coverability in VASS revisited: Improving Rackoff's bound to obtain conditional optimality. In *Proceedings of ICALP 2023*, volume 261 of *Leibniz International Proceedings in Informatics*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ICALP.2023.131.

33 Ranko Lazić and Sylvain Schmitz. The ideal view on Rackoff's coverability technique. *Information and Computation*, 277:104582, 2021. doi:10.1016/j.ic.2020.104582.

34 Jérôme Leroux. The reachability problem for Petri nets is not primitive recursive. In *Proceedings of FOCS 2021*, pages 1241–1252. IEEE Computer Society, 2022. doi:10.1109/FOCS52979.2021.00121.

35 Jérôme Leroux and Sylvain Schmitz. Reachability in vector addition systems is primitive-recursive in fixed dimension. In *Proceedings of LICS 2019*, pages 1–13. IEEE Computer Society, 2019. doi:10.1109/LICS.2019.8785796.

36 Richard J. Lipton. The reachability problem requires exponential space. Technical Report 62, Department of Computer Science, Yale University, January 1976. URL: http://www.cs.yale.edu/publications/techreports/tr63.pdf.

37 Stephen R. Mahaney. Sparse complete sets for NP: Solution of a conjecture of Berman and Hartmanis. *Journal of Computer and System Sciences*, 25(2):130–143, 1982. doi:10.1016/0022-0000(82)90002-2.

38 Carl A. Petri. *Kommunikation mit Automaten*. PhD thesis, Universität Bonn, 1962. URL: http://edoc.sub.uni-hamburg.de/informatik/volltexte/2011/160/.

39 M. Praveen. Small vertex cover makes Petri net coverability and boundedness easier. *Algorithmica*, 65(4):713–753, 2013. doi:10.1007/S00453-012-9687-6.

40 Charles Rackoff. The covering and boundedness problems for vector addition systems. *Theoretical Computer Science*, 6:223–231, 1978. doi:10.1016/0304-3975(78)90036-1.

41 Louis E. Rosier and Hsu-Chun Yen. A multiparameter analysis of the boundedness problem for vector addition systems. *Journal of Computer and System Sciences*, 32(1):105–135, 1986. doi:10.1016/0022-0000(86)90006-1.

42 Sylvain Schmitz. Automata column: The complexity of reachability in vector addition systems. *ACM SIGLOG News*, 3(1):3–21, 2016. doi:10.1145/2893582.2893585.

43 Sylvain Schmitz. Complexity hierarchies beyond ELEMENTARY. *ACM Transactions on Computation Theory*, 8(1), 2016. doi:10.1145/2858784.

44 Sylvain Schmitz and Lia Schütze. On the length of strongly monotone descending chains over $\mathbb{N}^d$. In *Proceedings of ICALP 2024*, volume 297 of *Leibniz International Proceedings in Informatics*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.ICALP.2024.153.

45 Dimitri Watel, Marc-Antoine Weisser, and Dominique Barth. Parameterized complexity and approximability of coverability problems in weighted Petri nets. In *Proceedings of Petri Nets 2017*, volume 10258 of *Lecture Notes in Computer Science*, pages 330–349. Springer, 2017. doi:10.1007/978-3-319-57861-3_19.

46 Michael Wehar. *On the Complexity of Intersection Non-Emptiness Problems*. PhD thesis, University at Buffalo, State University of New York, USA, 2016. URL: http://www.michaelwehar.com/documents/mwehar_dissertation.pdf.

A Details for Section 3

A.1 Proof of Claim 3.4

We give here a formal verification of the correctness of the reduction; that is, we prove Claim 3.4, recalled below for convenience.

▷ Claim 3.4. The following conditions are equivalent.

- There exists a word over Σ that is accepted by all the automata $\mathcal{D}_1, \ldots, \mathcal{D}_k$.
- There is run of $\mathcal{V}$ from configuration $p(\boldsymbol{u})$ to some configuration with state q.

Proof. We verify the two implications in order.

First direction. We first argue that if there is a word in accepted all the automata $\mathcal{D}_1, \ldots, \mathcal{D}_k$, then there exists a run in $\mathcal{V}$ from $p(\boldsymbol{u})$ to $q(\boldsymbol{v})$ for some $\boldsymbol{v} \geqslant \mathbf{0}$. In fact, we will show that there is a run from $p(\boldsymbol{u})$ to $q(\mathbf{0})$.

Suppose $w \in \Sigma^*$ is a word that is accepted by $\mathcal{D}_i$ for every $i \in \{1, \ldots, k\}$. We shall use vector-indexing notation $(w[1], w[2], \ldots)$ to refer to the letters that comprise w. We construct a run of $\mathcal{V}$ starting from $p(\boldsymbol{u})$ as follows. First, for each consecutive $j \in \{1, \ldots, |w|\}$, the run proceed through the $w[j]$-section as follows. Suppose that after the first $j-1$ letters have been read, the current state of $\mathcal{D}_i$ is $q_{i,a}$ (for some $1 \leqslant a \leqslant s$). Since w is accepted by every automaton $\mathcal{D}_i$, there is some $w[j]$-transition in $\mathcal{D}_i$ that is taken; suppose it is the transition $t_i = (q_{i,a}, w[j], q_{i,b})$. Then when the run traverses the $w[j]$-section from p back to p by using the pair of transitions that go through the intermediate state q_{t_i}, for each $i \in \{1, \ldots, k\}$. It is possible to indeed take these transitions because if $\mathcal{D}_i$ was in state $q_{i,a}$, then $\mathsf{x}_\mathsf{i} = a$ and $\mathsf{y}_\mathsf{i} = s - a$. Then, after t_i is taken in $\mathcal{D}_i$, the current state becomes $q_{i,b}$ which is reflected in the run in $\mathcal{V}$ since, after the second transition in the pair is taken, $\mathsf{x}_\mathsf{i} = b$ and $\mathsf{y}_\mathsf{i} = s - b$ is achieved.

Lastly, once every letter $w[1], \ldots, w[|w|]$ has been read, we know that the current state of $\mathcal{D}_i$ is a final state, for every i. Precisely, suppose that the current state of each automaton $\mathcal{D}_i$ is q_{i,f_i}, for some $1 \leqslant f_i \leqslant s$. Thus, in $\mathcal{V}$, the configuration $p(\boldsymbol{z})$ that is reached has the vector of counter values $\boldsymbol{z}$ for which $\mathsf{x_i} = f_i$ and $\mathsf{x_i} = s - f_i$. This means that in each of the parts of the checking section, a transition can be taken to subtract f_i from $\mathsf{x_i}$ and $s - f_i$ from $\mathsf{y_i}$. Thus, the constructed run can traverse the checking section and at the end reach configuration $q(\mathbf{0})$, as required.

Second direction. We now prove that if there is a run in $\mathcal{V}$ from $p(\boldsymbol{u})$ to $q(\boldsymbol{v})$ for some $\boldsymbol{v} \geqslant \mathbf{0}$, then there exists a word that is accepted by each automaton $\mathcal{D}_i$, for $i \in \{1, \ldots, k\}$.

Let π be a path such that $p(\boldsymbol{u}) \xrightarrow{\pi} q(\boldsymbol{v})$. We may split π into subpaths $\pi_1, \ldots, \pi_m, \pi_{m+1}$ such that for every $j \in \{1, \ldots, m\}$, π_j is a path from p back to p that only visits p at the start and the end, and π_{m+1} is a path from p to q. Due to the overall structure of $\mathcal{V}$ (see Figure 3), we know that for all $j \in \{1, \ldots, m\}$, π_j is a path inside a σ_j-section, for some $\sigma_j \in \Sigma$, whereas π_{m+1} traverses the checking section. We will argue that the word

$$w \stackrel{\text{def}}{=} \sigma_1 \, \sigma_2 \, \cdots \, \sigma_m$$

is accepted by every automaton $\mathcal{D}_i$, for $i \in \{1, \ldots, k\}$.

For every $j \in \{1, \ldots, m\}$, let $p(\boldsymbol{u_j})$ be the intermediate configuration between π_j and π_{j+1}. That is, we have:

$$p(\boldsymbol{u}) \xrightarrow{\pi_1} p(\boldsymbol{u_1}) \xrightarrow{\pi_2} p(\boldsymbol{u_2}) \to \cdots \to p(\boldsymbol{u_{m-1}}) \xrightarrow{\pi_m} p(\boldsymbol{u_m}) \xrightarrow{\pi_{m+1}} q(\boldsymbol{v}).$$

Initially, in $\boldsymbol{u}$, we know that for all $i \in \{1, \ldots, k\}$, the invariant $\mathsf{x_i} + \mathsf{y_i} = s$ and $\mathsf{x_i} \geqslant 1$ holds, and thus $\mathsf{x_i}$ contains the index of a state of $\mathcal{D}_i$. By the design of $\mathcal{V}$, we deduce that this invariant holds true also for all intermediate configurations $p(\boldsymbol{u_1}), p(\boldsymbol{u_2}), \ldots, p(\boldsymbol{u_m})$.

Now, consider the sub-run $p(\boldsymbol{u_j}) \xrightarrow{\pi_j} p(\boldsymbol{u_{j+1}})$. In order to leave state p, go through the σ_j-section, and return to p, it must be the case that each part is successfully passed. Suppose that at in $\boldsymbol{u_j}$, it is true that $\mathsf{x_i} = a$ (and, due to the invariant, we know that $\mathsf{y_i} = s - a$). In order to pass the i-th part, it must be true that there must be a transition, say t, that subtracts a from $\mathsf{x_i}$ and $s - a$ from $\mathsf{y_i}$. This is true because all transitions in the i-th part subtract x from $\mathsf{x_i}$ and $s - x$ from $\mathsf{y_i}$ for some $1 \leqslant x \leqslant s$. Inside the i-th part of the σ_j-section, after t is taken, there is a following transition, say t', that adds b to $\mathsf{x_i}$ and $s - b$ to $\mathsf{y_i}$ for some $1 \leqslant b \leqslant s$. Now since t and t' are present in i-th part of the $\sigma_{f(j)}$-section, then we know there is a transition $(q_{i,a}, \sigma_{f(j)}, q_{i,b})$ in $\mathcal{D}_i$. Indeed, the run taking t, t' is simulating $\mathcal{D}_i$ using $(q_{i,a}, \sigma_j, q_{i,b})$ (and so $\mathcal{D}_i$ has now read the j-th letter of the word). A straightforward induction now shows that for each $j \in \{1, \ldots, m\}$, vector $\boldsymbol{u}_j$ encodes, in its counters, the states in which the automata $\mathcal{D}_1, \ldots, \mathcal{D}_k$ are after reading the first j letters of w.

We conclude by analysing the final sub-run $p(\boldsymbol{u_m}) \xrightarrow{\pi_m} q(\boldsymbol{v})$, and arguing that the state in which each automaton $\mathcal{D}_i$ is after reading w must be a final state. This holds for the same reason used before: the transition that subtracts a from $\mathsf{x_i}$ and $s - a$ from $\mathsf{y_i}$ in the i-th part of the checking section is only present when $q_{i,a}$ is a final state of $\mathcal{D}_i$. Thus, in order to reach q, one transition from each part of the checking section must be taken, which can only be true when the current state of automaton $\mathcal{D}_i$ is a final state. Thus, the word $w = \sigma_1 \, \sigma_2 \, \ldots \, \sigma_m$ is accepted by all the automata $\mathcal{D}_1, \ldots, \mathcal{D}_k$. ◁

Exact Algorithms and Hardness Result for the Boolean Connectivity Problem of k-Horn Formulas

Takashi Horiyama
Faculty of Information Science and Technology, Hokkaido University, Sapporo, Japan

Yuto Okura
Graduate School of Information Science and Technology, Hokkaido University, Sapporo, Japan

Kazuhisa Seto
Faculty of Information Science and Technology, Hokkaido University, Sapporo, Japan

Junichi Teruyama
Graduate School of Information Science, University of Hyogo, Kobe, Japan

Abstract

The Boolean connectivity problem asks whether the set of satisfying assignments of a given Boolean formula forms a connected subgraph in the n-dimensional hypercube. This problem is known to be coNP-complete, even when restricted to k-Horn formulas for $k \geq 3$, as shown by Makino, Tamaki, and Yamamoto. In this paper, we further investigate the complexity of the Boolean connectivity problem for k-Horn formulas, referred to as Conn k-Horn. We first present an exact exponential-time algorithm for Conn k-Horn without any structural restrictions. Our algorithm builds on the deterministic PPZ algorithm proposed by Paturi, Pudlák, and Zane. It runs in $O^*(2^{(1-1/2k)n})$ time, achieving an exponential improvement over the previously known algorithm for the Boolean connectivity problem of k-CNF formulas, shown by Makino, Tamaki, and Yamamoto. We then examine both algorithmic and hardness results for Conn 3-Horn under bounded variable occurrences. On the algorithmic side, we propose a polynomial-time algorithm for Conn 3-Horn when each clause contains exactly three literals and each variable appears at most three times. This result generalizes to Conn k-Horn under the same structural constraints, in which each clause contains exactly k literals and each variable appears at most k times. On the hardness side, we prove that Conn 3-Horn remains coNP-complete even when restricted to instances in which each variable appears exactly four times.

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

Keywords and phrases k-Horn, Boolean connectivity, bounded variable occurrence, hardness, exact algorithm, satisfiability

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.25

Funding *Takashi Horiyama*: JSPS KAKENHI Grant Number 20H05964 and 23K24806
Kazuhisa Seto: JSPS KAKENHI Grant Number 22H00513, 23K24806 and 24K02898
Junichi Teruyama: JSPS KAKENHI Grant Number 22K11910, 23K28039, 23K28040

1 Introduction

The Boolean connectivity problem asks whether the satisfying assignments of a given Boolean formula are connected over the n-dimensional hypercube. The structure and connectivity of the satisfying assignments are related to analyzing the satisfiability algorithm and the threshold of satisfiability. The satisfiability problem is characterized by the famous Schaefer dichotomy theorem [12]. Schaefer's class denotes the class of formulas, e.g., 2-CNF, (dual) Horn, and affine formulas. Ekin, Hammer, and Kogan [5] showed that the Boolean connectivity of DNF is solvable in time linear in the size of DNF, while determining the connectivity of falsifying assignments of DNF is coNP-hard, i.e., the Boolean connectivity of CNF is

© Takashi Horiyama, Yuto Okura, Kazuhisa Seto, and Junichi Teruyama;
licensed under Creative Commons License CC-BY 4.0

coNP-hard. Gopalan, Kolaitis, Maneva, and Papadimitriou [6] deeply investigated the computational complexity of the Boolean connectivity problem. They showed that it is either coNP-complete or PSPACE-complete for non-Schaefer's class, and in coNP for Schaefer's class. They also conjectured that it is solvable in polynomial time for Schaefer's class. Makino, Tamaki, and Yamamoto [7] and Schwerdtfeger [15] gave a negative answer by proving the coNP-completeness of the Boolean connectivity problem for k-Horn and other related formulas.

Although the trichotomy theorem of the Boolean connectivity problem has been fully established, only a few algorithms are known for instances beyond P. For the connectivity of k-CNF formulas (Conn k-CNF), it is solvable in polynomial time when $k = 2$ [6]. Makino et al. [8] presented a moderately exponential-time algorithm for Conn k-CNF ($k \geq 3$) over n variables and m clauses which is PSPACE-complete. Their algorithm constructs the solution graph over partial satisfying assignments and then determines the connectivity of the graph. It runs in $O^*(2^{(1-c_k)n})$ time and exponential space, where $c_k = \log \beta_k/(1 + \log \beta_k)$ and β_k (< 2) is the largest positive real number that satisfies $x^k - x^{k-1} - \cdots - x - 1 = 0$. The O^* notation suppresses polynomial factors in n and m. Throughout this paper, the base of the logarithm is 2. Note that $c_k = 1/2^{O(k)}$.

In this paper, we deeply investigate the complexity of the Boolean connectivity of k-Horn formulas (Conn k-Horn). We first consider an exact algorithm for Conn k-Horn without any structural restrictions. The algorithm of Makino et al. can also solve Conn k-Horn in the same running time since k-Horn formulas are subsets of k-CNF formulas. Hence, it is a natural question whether there exists a faster algorithm for Conn k-Horn. Makino et al. [7] presented an exact algorithm in polynomial time in terms of the number of variables and the size of the characteristic set of a given Horn formula. Unfortunately, the efficiency of finding the characteristic set of a given Horn formula is not yet well-known. We give an affirmative answer by utilizing the deterministic PPZ algorithm for k-SAT shown by Paturi, Pudlák, and Zane [10]. The deterministic PPZ algorithm can find not only a satisfiability assignment but also all locally minimal satisfying assignments. Moreover, Conn k-Horn can be solved to find a locally minimal satisfying assignment with Hamming weight at least one [6]. These help us solve Conn k-Horn exponentially faster than the algorithm of Conn k-CNF.

▶ **Theorem 1.** *Given a k-Horn formula over n variables, there exists a deterministic $O^*(2^{(1-1/2k)n})$ time and polynomial-space algorithm for* Conn k-Horn.

The slightly modified algorithm above can count the number of connected components in the solution graph of a given k-Horn formula.

▶ **Corollary 2.** *Counting the number of connected components in the solution graph of a given k-Horn formula over n variables can be done in deterministic $O^*(2^{(1-1/2k)n})$ time and $O^*(2^{(1-1/2k)n})$ space.*

We next consider Conn 3-Horn with a bounded number of occurrences of each variable. We present both algorithmic and hardness results.

▶ **Theorem 3.** Conn 3-Horn *is solvable in polynomial time when each clause has length exactly three and each variable appears at most three times.*

The key idea of our algorithm is based on the relationship between the solution graph of a Horn formula and a non-empty maximal self-implicating set, as shown in [15]. Any variable in a self-implicating set is forced to be true when all the other variables in the set are

true. The solution graph is disconnected if and only if there exists a non-empty maximal self-implicating set and no clauses with only negative literals of these variables. We show that such a set can be found in polynomial time.

▶ **Theorem 4.** Conn 3-Horn *is* coNP*-complete even when each variable appears exactly four times.*

We first prove the complement of Conn 3-Horn remains NP-complete even if each variable appears at most four times by a polynomial-time reduction from Monotone Not-All-Equal 3-SAT with each variable appearing exactly four times. This problem is known to be NP-complete [3]. We next show Theorem 4 by providing a polynomial-time reduction from the above problem.

Related Work

The connectivity problem between two satisfying assignments (st-Conn) has also been extensively studied. Gopalan et al. [6] showed that st-Conn is solvable in polynomial time for Schaefer's and a part of non-Schaefer's class and is PSPACE-complete otherwise. Scharpfenecker [13] proved that the complexity of SAT is equivalent to that of st-Conn in Schaefer's class. This implies that st-Conn is P-complete for Horn formulas and is NL-complete for 2-CNFs. Cardinal, Demaine, Eppstein, Hearn, and Winslow [2] showed that st-Conn of Planar Monotone Not-All-Equal 3-SAT is PSPACE-complete.

The problem of finding the shortest path in the solution graph of Boolean formulas has also been investigated. Mouawad, Nishimura, Pathak, and Raman [9] investigated the complexity of this problem. They established a trichotomy theorem stating that the problem is either in P, NP-complete, or PSPACE-complete. They also showed that some classes of Boolean formulas exist where the shortest path can be found in polynomial time, even though its length is not equal to the symmetric difference of the values of each variable in s and t. Bonsma, Mouawad, Nishimura, and Raman [1] showed the problem is W[1]-hard when parameterized by the path length ℓ from s to t.

The complexity of the isomorphism of two solution graphs was studied in [14]. This problem is PSPACE-hard and lies in EXP for general formulas, and is $\mathsf{C}_{=}$P-complete for 2-CNF. The class $\mathsf{C}_{=}$P is a subclass of PSPACE and is defined in terms of exact counting.

2 Preliminaries

Let x be a Boolean variable that takes true (1) or false (0). A *literal* is a Boolean variable x (positive literal) or its negation $\bar{x}$ (negative literal). A *clause* is a disjunction of literals. The *length* of clause C is defined as the number of literals in C. A *unit clause* is a clause of length one. A conjunctive normal form (CNF) formula is a conjunction of clauses, and a k-CNF formula is a CNF formula if the length of each clause is at most k. A CNF formula is *monotone* if all clauses in the formula consist of only positive (or negative) literals. In this paper, we use Monotone CNF as CNF formulas with all positive literals unless otherwise stated. A *Horn clause* is a clause including at most one positive literal. A Horn formula is a CNF formula whose all clauses are Horn, and a k-Horn formula is a k-CNF formula and a Horn formula. Let $X = \{x_1, x_2, ..., x_n\}$ be a set of Boolean variables. An *assignment* of a Boolean formula φ over X is $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n) \in \{0,1\}^n$ such that $x_i = \alpha_i$ for all i. A *partial assignment* of a Boolean formula φ over X is $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n) \in \{0,1,*\}^n$ such that $x_i = \alpha_i$ for all i. A partial assignment with $\alpha_i = *$ means variable x_i is not assigned a 0/1 value. For a variable x and an assignment α, we denote by $\alpha(x)$ the value of x under

an assignment α. We can simplify a Boolean formula φ by a partial assignment α in a natural way, and the resulting formula is denoted by $\varphi|_\alpha$. An (partial) assignment α is a *(partial) satisfying assignment* of a Boolean formula φ if $\varphi|_\alpha$ is true. The satisfiability problem is to determine whether there exists a satisfying assignment of a given Boolean formula φ.

Many excellent exponential-time algorithms for the satisfiability problem of k-CNF formulas (k-CNF SAT) are known. One of the famous algorithms is shown by Paturi, Pudlák, and Zane [10].

▶ **Theorem 5** ([10]). *There exists an algorithm that can solve* k-CNF SAT *over* n *variables in a deterministic* $O^*(2^{(1-1/2k)n})$ *time and polynomial space.*

For an assignment $\alpha \in \{0,1\}^n$, the *Hamming weight* of α is $|\{i : \alpha_i = 1 \ (1 \leq i \leq n)\}|$. For two assignments $\alpha, \alpha' \in \{0,1\}^n$, the *Hamming distance* between α and α', denoted by $d(\alpha, \alpha')$, is $|\{i : \alpha_i \neq \alpha'_i \ (1 \leq i \leq n)\}|$. We define the *solution graph* of a Boolean formula φ as $G_\varphi := (V_\varphi, E_\varphi)$, where V_φ is a set of all satisfying assignments of φ, that is, $V_\varphi = \{\alpha : \varphi|_\alpha = 1, \alpha \in \{0,1\}^n\}$, and $E_\varphi = \{(\alpha, \alpha') : d(\alpha, \alpha') = 1 \text{ and } \alpha, \alpha' \in V_\varphi\}$.

▶ **Definition 6.** *The Boolean connectivity problem* (Conn) *is to ask whether the solution graph* G_φ *of a given Boolean formula* φ *over* n *variables is connected.*

Makino, Tamaki, and Yamamoto [7] showed that Conn 3-Horn is coNP-complete, thus the following lemma holds.

▶ **Lemma 7** ([7]). Conn k-Horn *is* coNP*-complete when* $k \geq 3$.

Given two assignments $\alpha, \alpha' \in \{0,1\}^n$, we denote $\alpha \leq \alpha'$ as the coordinate-wise partial order if $\alpha_i \leq \alpha'_i$ for each $1 \leq i \leq n$. A satisfying assignment α for a given formula φ is *locally minimal* if α has no neighboring satisfying assignment α' with $d(\alpha, \alpha') = 1$ and $\alpha' \leq \alpha$. For $v, v' \in V_\varphi$, a *monotone path* from v to v' is a path in G_φ, $v \to u_1 \to \cdots \to u_r \to v'$ such that $v \leq u_1 \leq \cdots \leq u_r \leq v'$. Gopalan, Kolaitis, Maneva, and Papadimitriou [6] showed a notable connection between locally minimal satisfying assignments and the connectivity of Horn formulas.

▶ **Lemma 8** ([6]). *Given a Horn formula* φ*, every connected component of* G_φ *has a unique locally minimal satisfying assignment. Moreover, there exists a monotone path from the locally minimal satisfying assignment to each satisfying assignment in the same connected component.*

The following corollary follows from that any Horn formula without unit clauses over n variables has an all-zero satisfying assignment (0^n) and Lemma 8.

▶ **Corollary 9** ([7]). *Given a Horn formula* φ *without unit clauses, the solution graph* G_φ *is connected if and only if no locally minimal non-zero satisfying assignment exists.*

Let $P(C)$ (resp. $N(C)$) denote the set of variables that appear in positive (resp. negative) literals in a clause C of φ. Makino et al. [7] introduced the following Boolean formula Φ_φ.

$$\Phi_\varphi = \varphi \wedge \bigwedge_{i=1}^{n} D_i, \tag{1}$$

$$D_i = \overline{x_i} \vee \bigvee_{C \in \varphi : P(C) = \{x_i\}} \bigwedge_{y \in N(C)} y.$$

For example, given a Horn formula

$$\varphi = (x_1 \vee \overline{x_2} \vee \overline{x_3})(x_1 \vee \overline{x_3} \vee \overline{x_4})(\overline{x_2} \vee \overline{x_3} \vee \overline{x_4})(\overline{x_2} \vee x_3 \vee \overline{x_4})(\overline{x_1} \vee \overline{x_3} \vee x_4),$$

then

$$D_1 = \overline{x_1} \vee x_2x_3 \vee x_3x_4,\ D_2 = \overline{x_2},\ D_3 = \overline{x_3} \vee x_2x_4,\ D_4 = \overline{x_4} \vee x_1x_3.$$

Note that each D_i is either a unit clause with a negative literal or a conjunction of one negative literal and disjunctions of positive literals. The satisfiability of Φ_φ is strongly related to the connectivity of the solution graph of a Horn formula φ because a satisfying assignment of Φ_φ is a locally minimal satisfying assignment of φ.

▶ **Lemma 10** ([7]). *Given a Horn formula φ without unit clauses over n variables, the solution graph G_φ is disconnected if and only if Φ_φ has a non-zero satisfying assignment.*

A *restraint clause* is a clause with only negative literals, and a *restraint set* is a set of variables included in restraint clauses. An *implication clause* is a clause that has one positive literal and one or more negative literals. A Boolean variable x is said to be *implied* by a set of variables U if setting all variables in U to true forces x to be true. A *self-implicating set* U is a set of variables, where every $x \in U$ is implied by $U \setminus \{x\}$. We also say that U is a *maximal self-implicating set* if U is a set of all variables implied by U. For maximal self-implicating sets, there exists a relation between locally minimal satisfying assignments and maximal self-implicating sets.

▶ **Lemma 11** ([15]). *For every Horn formula φ without positive unit clauses, there is a bijection correlating each connected component φ_i with a maximal self-implicating set U_i containing no restraint set, i.e., U_i consists of the variables assigned true in the minimum assignment of φ_i (the 'lowest' component is correlated with the empty set).*

The following corollary follows from Lemma 11 and Corollary 9.

▶ **Corollary 12** ([15]). *The solution graph of a Horn formula φ without positive unit clauses is disconnected if and only if φ has a non-empty maximal self-implicating set containing no restraint set.*

3 Exact Algorithm for the Boolean Connectivity of k-Horn formulas based on the deterministic PPZ Algorithm

In this section, we present a faster algorithm for Conn k-Horn based on the deterministic PPZ algorithm [10]. Before presenting our new algorithm, we describe the notations and behaviors of the deterministic PPZ algorithm, referred to as DetPPZ. Let φ be a k-CNF formula with n variables and m clauses, and let α be a satisfying assignment. A clause $C_{(\alpha,i)}$ is a *critical clause* for the variable x_i at the solution α if $C_{(\alpha,i)}$ is to be false by flipping α_i. A variable x_i is a *critical variable* if there exists a critical clause $C_{(\alpha,i)}$. The DetPPZ first checks whether there exists a satisfying assignment of φ to check all assignments with Hamming weight at most ϵn where $0 < \epsilon < 1/2$. If there is no such satisfying assignment, the DetPPZ tries to find a locally minimal satisfying assignment α with Hamming weight at least ϵn. Note that φ is falsified by any assignment obtained by flipping any 1 in α to 0. Thus, α must have at least ϵn critical variables. Let $\sigma = (\sigma_1, \ldots, \sigma_n)$ be a permutation of $\{1, \ldots, n\}$. The DetPPZ picks up a permutation σ in a "good" small sample space and branches by assigning a 0/1 value to x_{σ_i} according to the order in σ. If any critical

Algorithm 1 PPZ-Conn-kHorn(φ).

```
1  Construct the set of permutations S in Lemma 13
2  for all assignments α with at most ϵn 1's without all 0's do
3      if α satisfies φ and α is locally minimal then
4          return NO
5  for all permutations σ in S do
6      for all strings s of n(1 − ϵ/k) + 1 bits do
7          for i = 1 to n do
8              Let x_j be the i-th variable according to σ (i.e., j = σ_i).
9              if there exists a unit clause x_j or ¬x_j then
10                 set x_j to make that clause true
11             else if there exists an unused bit from s then
12                 set x_j equal to the next unused bit from s
13             else
14                 go to the next string in the for loop of lines 6–17.
15                                          ▷ there are no unused bits from s
16         if φ is satisfiable and the satisfying assignment is locally minimal then
17             return NO
18 return YES
```

variable appears last among the variables in the corresponding critical clause, then it can be assigned correctly. It helps to reduce the number of branches in the algorithm. Modifying the DetPPZ such that it does not halt even if it finds a satisfying assignment with Hamming weight at most ϵn, the DetPPZ can find all locally minimal satisfying assignments. From Corollary 9, Conn k-Horn can be solved by checking whether there exists some locally minimal satisfying assignment except for 0^n assignment. Thus, by combining the (modified) DetPPZ and the procedure checking locally minimality, we solve Conn k-Horn in the same running time as the DetPPZ. We describe our algorithm, PPZ-Conn-kHorn, in Algorithm 1.

The following "good" sample space of permutations using k-wise independent variables is useful.

▶ **Lemma 13** ([10]). *One can construct the set of permutations S of size $O(n^{3k})$ that satisfies the following condition: for any set Y of up to k variables, any variable $y \in Y$, and for a randomly chosen permutation from S, the probability that y appears last among the variables in Y is at least $1/|Y| - 1/n$.*

The following lemma immediately follows from the proof of correctness of the deterministic PPZ algorithm shown in [10]. We provide almost the same proof for self-containedness in this paper.

▶ **Lemma 14.** *Let S be the set of permutations constructed by Lemma 13. Let α be an arbitrary satisfying assignment with at least ϵn critical clauses. There exists a permutation in S that produces α.*

Proof. Let α be a satisfying assignment with at least ϵn critical clauses. We now choose a permutation σ from S at random. For each critical clause at α, the probability that the critical variable x appears last in σ among the variables in the clause is at least $1/k - 1/n$ by Lemma 13. Since the number of critical clauses is at least ϵn, the expected number of times this event appears is at least $\epsilon n(1/k - 1/n) = \epsilon n/k - \epsilon$, and there exists some σ in S that achieves at least the expectation. With respect to such σ, we assign at most $n - (\epsilon n/k - \epsilon) < n(1 - \epsilon/k) + 1$ variables to satisfy φ. Thus, by checking all strings of $n(1 - \epsilon/k) + 1$ bits, the permutation σ can produce a satisfying assignment α. ◀

We next show that the locally minimality of satisfying assignments can be efficiently checked.

▶ **Lemma 15.** *Let α be a satisfying assignment of φ with Hamming weight w. One can check whether α is locally minimal in $O(mw + n)$ time.*

Proof. From the definition of locally minimality, if any satisfying assignment α of φ is locally minimal, then any assignment α' obtained by flipping any 1's bit is not a satisfying assignment of φ. It can be done by checking whether α' satisfies φ. Since the Hamming weight of α is w, we can check the locally minimality of α in $O(mw + n)$ time. ◀

We present the main lemma, which establishes the correctness of our algorithm.

▶ **Lemma 16.** *The deterministic PPZ algorithm can find all locally minimal satisfying assignments of φ.*

Proof. The deterministic PPZ algorithm first checks all assignments with Hamming weight at most ϵn. Hence, any satisfying assignment α with Hamming weight at most ϵn can be found. We can check whether α is locally minimal from Lemma 15.

We next consider that any locally minimal satisfying assignment α has Hamming weight at least ϵn. Since α is a locally minimal satisfying assignment and has at least ϵn 1's, there exists at least ϵn critical clauses at α. From Lemma 14, α must be produced by some permutation in the sample space S constructed in the algorithm. The algorithm checks all permutations in S, then it can find α. We can check whether α is truly locally minimal from Lemma 15. ◀

We are now ready to prove Theorem 1.

▶ **Theorem 1.** *Given a k-Horn formula over n variables, there exists a deterministic $O^*(2^{(1-1/2k)n})$ time and polynomial-space algorithm for* Conn k-Horn.

Proof. We can solve Conn k-Horn by checking whether there exists some locally minimal non-zero satisfying assignment from Corollary 9. Lemma 16 shows that the deterministic PPZ algorithm can find all locally minimal satisfying assignments. Thus, PPZ-Conn-kHorn in Algorithm 1 solves Conn k-Horn correctly. The deterministic PPZ algorithm runs in $O^*(2^{(1-1/2k)n})$ time and polynomial space. Checking locally minimality can be done $O(mn)$ time for each satisfying assignment from Lemma 15. Thus, our algorithm runs in $O^*(2^{(1-1/2k)n})$ time and polynomial space. ◀

By adding the procedure of counting the number of distinct locally minimal satisfying assignments (see Algorithm 2), we obtain the following Corollary.

▶ **Corollary 2.** *Counting the number of connected components in the solution graph of a given k-Horn formula over n variables can be done in deterministic $O^*(2^{(1-1/2k)n})$ time and $O^*(2^{(1-1/2k)n})$ space.*

Algorithm 2 COUNT-CC-kHORN(φ).

```
1  count ← 0
2  Construct the set of permutations S in Lemma 13
3  for all assignments α with at most ϵn 1's do
4      if α satisfies φ, and α is locally minimal and has never been produced then
5          count ← count+1
6  for all permutations σ in S do
7      for all strings s of n(1 − ϵ/k) + 1 bits do
8          α ← 0^n
9          for i = 1 to n do
10             Let x_j be the i-th variable according to σ (i.e.,j = σ_i).
11             if there exists a unit clause x_j or ¬x_j then
12                 set α_j to make that clause true
13             else if there exists an unused bit from s then
14                 set α_j equal to the next unused bit from s
15             else
16                 go to the next string in the for loop of lines 6–17.
17                                                  ▷ there are no unused bits from s
18         if φ is satisfiable, and the satisfying assignment α is locally minimal and has
            never been produced then
19             count ← count+1
20 return count
```

4 Complexity on the Boolean Connectivity of 3-Horn Formulas with Bounded Variable Occurrence

In this section, we consider the Boolean connectivity of 3-Horn, whose variable occurrence is bounded. Section 4.1 presents a deterministic polynomial-time algorithm for CONN 3-HORN with each clause of length exactly three and each variable appearing at most three times (CONN E3-HORN-3). Section 4.2 shows the coNP-complete of CONN 3-HORN where each variable appears exactly four times (CONN 3-HORN-E4).

4.1 Polynomial-time Algorithm

We present a deterministic polynomial-time algorithm for CONN E3-HORN-3. Our algorithm can solve the connectivity of k-Horn formulas with the length of each clause exactly k and each variable appearing at most k (CONN Ek-HORN-k). For a set of variables X and a CNF formula φ, we denote by $C_{\varphi[X]}$ the set of all clauses in φ that consist of only variables in X.

▶ **Lemma 17.** *Let φ be an instance of* CONN E3-HORN-3 *over $X = \{x_1, x_2, \ldots, x_n\}$. The solution graph G_φ is disconnected if and only if there exists a partition of the set X into two sets U and $X \setminus U$ and a partition of the set $C_{\varphi[X]}$ of clauses of φ into two sets $C_{\varphi[U]}$ and $C_{\varphi[X\setminus U]}$ such that U is a non-empty set of variables which appear once as a positive literal.*

Proof. We first prove the if-part, and assume that a variable set U satisfies the if-condition. We show that U is a non-empty maximal self-implicating set containing no restraint set, then G_φ is disconnected from Corollary 12. The number of implication clauses in $C_{\varphi[U]}$ is $|U|$ since each variable in U appears once as a positive literal. These implication clauses have $3|U|$ literals since the length of each clause in $C_{\varphi[U]}$ is three, and then the total number of literals of clauses in $C_{\varphi[U]}$ is at least $3|U|$. However, the total number of occurrences of variables in U is at most $3|U|$ since each variable appears at most three times. These lead to the total number of literals in $C_{\varphi[U]}$ being exactly $3|U|$. Thus, all literals in $C_{\varphi[U]}$ appear in implication clauses in $C_{\varphi[U]}$, i.e., there is no restraint set in U. For any implication clause C, the positive literal x in C is assigned true when the other negative literal(s) are assigned false. This implies that every $x \in U$ is implied by $U \setminus \{x\}$. In addition, any variable in any clause in $C_{\varphi[X\setminus U]}$ cannot be implied by U since $U \cap (X \setminus U) = \emptyset$. Hence, U is a non-empty maximal self-implicating set and contains no restraint set. This concludes the proof of the if-part.

We next prove the only-if part. Assume that G_φ is disconnected. From Corollary 12, φ has a non-empty maximal self-implicating set U containing no restraint set. This implies that every variable in U appears as a positive literal at least once. We claim that the number of implication clauses with only variables in U is $|U|$. There exists at least $|U|$ implication clauses in $C_{\varphi[X]}$ since U is a self-implicating set, i.e., there exists a clause $u \vee \overline{x} \vee \overline{y}$ for every $u \in U$. Since the total number of occurrences of variables in U is at most $3|U|$ and the length of each clause is three, there exist at most $|U|$ implication clauses with only variables in U. If the number of such implication clauses with only variables in U is at most $|U| - 1$, there exists some $u \in U$ appearing in an implication clause as positive literals with at least one variable x in $X \setminus U$. We denote $(u \vee \overline{x} \vee \overline{y})$ as such a clause. In this case, u is implied by x and y. This contradicts the fact that U is a maximal self-implicating set. Thus, the number of implication clauses with only variables in U is $|U|$, and this implies that every variable in U appears as a positive literal only once. The number of literals of these clauses is $3|U|$, and it equals the maximum total number of occurrences of variables in U. This implies that the pair $(U, X \setminus U)$ is a partition of X, and the pair of $(C_{\varphi[U]}, C_{\varphi[X\setminus U]})$ is a partition of $C_{\varphi[X]}$. This concludes the proof of the only-if part. ◀

Now, we present a polynomial-time algorithm based on Lemma 17.

▶ **Theorem 3.** Conn 3-Horn *is solvable in polynomial time when each clause has length exactly three and each variable appears at most three times.*

Proof. A given 3-Horn formula φ over $X = \{x_1, x_2, \ldots, x_n\}$ has at most $3n$ literals and n clauses since the length of each clause is three and each variable appears at most three times. We find a non-empty variable set that satisfies Lemma 17 to determine the connectivity of the solution graph of φ.

We first enumerate the non-empty variable set U such that $C_{\varphi[X]} = C_{\varphi[U]} \cup C_{\varphi[X\setminus U]}$ and $C_{\varphi[U]} \cap C_{\varphi[X\setminus U]} = \emptyset$ as follows. We construct the graph $G = (X, E)$, where $E = \{(x_i, x_j) : C_{\varphi[X]}$ has a clause containing $x_i, x_j \in X\}$. It is easy to see that the variable set U' corresponding to the vertices in any connected component of G constructs $C_{\varphi[U']}$. Thus, we can enumerate the non-empty variable sets to enumerate the connected components of G. We use the depth-first search for G to enumerate all the connected components. It remains to check whether each U' is a set of variables that appear once as a positive literal. If at least one such U' exists, the solution graph is disconnected. Otherwise, the solution graph is connected, as shown in Lemma 17.

We now estimate the running time of this algorithm. Constructing the graph G can be done in $O(n)$ time since the number of edges is at most $3n$. Enumerating the non-empty variable set U' can be done in time $O(n)$ by the depth-first search. Note that the number of non-empty variable sets is $O(n)$. We can check whether each U' is a set of variables that appear once as a positive literal in $O(n)$ by checking the clauses. Hence, our algorithm runs in time $O(n^2)$. ◀

We can show Corollary 18 since the discussion in Lemma 17 also holds for $k \geq 3$.

▶ **Corollary 18.** *Let φ be an instance of* Conn Ek-Horn-k *over $X = \{x_1, x_2, \ldots, x_n\}$. The solution graph G_φ is disconnected if and only if there exists a partition of the set X into two sets U and $X \setminus U$ and a partition of the set $C_{\varphi[X]}$ of clauses of φ into two sets $C_{\varphi[U]}$ and $C_{\varphi[X\setminus U]}$ such that U is a non-empty set of variables which appear once as a positive literal.*

Gopalan et al. [6] showed that Conn 2-CNF is solvable in polynomial time with no bounded variable occurrence. The proof of Theorem 3 also holds for $k \geq 3$ from Corollary 18. Therefore, we can show Corollary 19.

▶ **Corollary 19.** *For $k \geq 2$,* Conn Ek-Horn-k *is solvable in polynomial time.*

We can show the following corollary from Corollary 18.

▶ **Corollary 20.** *For $k \geq 3$, given a k-Horn formula φ, where each clause length and each variable occurrence are exactly k, the number of connected components in the solution graph of φ is at most $2^{\lfloor n/k \rfloor}$.*

Proof. Let φ be an instance of Conn k-Horn with each clause length exactly k and each variable occurrence exactly k, where the solution graph of φ is disconnected. A variable set of φ is denoted by X. Then, there exists at least one variable set $U \subseteq X$ such that $C_{\varphi[U]}$ and $C_{\varphi[X\setminus U]}$ partition $C_{\varphi[X]}$ and U is a non-empty set of variables that appear once as a positive literal. Now, we assume that φ has m variable sets U_i $(1 \leq i \leq m)$ and a variable set V such that $C_{\varphi[X]} = \bigcup_{i=1}^{m} C_{\varphi[U_i]} \cup C_{\varphi[V]}$ and $\bigcup_{i=1}^{m} U_i \cup V = X, U_i \cap V = \emptyset$, and $U_i \cap U_j = \emptyset$ for any i, j $(i \neq j)$, where U_i is a non-empty set of variables which appear once as a positive literal. Then, we can construct all locally minimal satisfying assignments by setting the value 0 or 1 to all variables for each U_i $(1 \leq i \leq m)$ and setting the value 0 to all variables in V from Lemma 11. Therefore, there exist 2^m locally minimal satisfying assignments of φ. Moreover, m is a multiple of k since the length of each clause and the number of occurrences of each variable are exactly k. Hence, the maximum value of m is $\lfloor n/k \rfloor$. ◀

4.2 Hardness

We prove the coNP-completeness of Conn 3-Horn where each variable appears at most four times (Conn 3-Horn-4), then we prove the coNP-completeness of Conn 3-Horn-E4 by a polynomial-time reduction from it. Instead of proving the coNP-completeness of Conn 3-Horn-4, we show the NP-completeness of the complement problem of Conn 3-Horn-4 (DisConn 3-Horn-4) that asks whether the solution graph of a given 3-Horn formula with each variable appearing at most four times is disconnected. The proof is based on a polynomial-time reduction from Monotone Not-All-Equal E3-SAT-E4 (for short, Monotone NAE E3-SAT-E4) by utilizing the framework of Makino et al. [7]. An *NAE-satisfying assignment* is a satisfying assignment in which each clause has at least one literal assigned false and at least one literal assigned true. Given a Monotone 3-CNF formula

ϕ with n variables and m clauses in which each clause has exactly three literals and each variable appears exactly four times, Monotone NAE E3-SAT-E4 asks whether there exists an NAE-satisfying assignment of ϕ. It is known that this problem is NP-complete [3].

We first show the construction of a 3-Horn formula from an instance of Monotone NAE E3-SAT-E4. Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of variables. Let ϕ be an instance of Monotone NAE E3-SAT-E4 over X with m clauses. Let $C_\phi = \{C_1, C_2, \ldots, C_m\}$ be the set of clauses of the formula φ. For simplicity, if the variable x_j appears in the clause C_i, we write $x_j \in C_i$. We first replace the k-th occurrence of $x_i \in X$ with a new variable $x_{i,k}$ and we denote by X_φ the set of new variables created from X. Let $Y = \{y_i : 1 \le i \le m\}$, $Z = \{z_i : 1 \le i \le m-2\}$ and $Q = \{q_{i,j} : 1 \le i \le n, 1 \le j \le 4\} \cup \{q_0, q', q_{n+1,1}\}$. We construct the formula φ_ϕ over $X_\varphi \cup Y \cup Z \cup Q$ from ϕ as follows.

$$
\begin{aligned}
\varphi_\phi \stackrel{\text{def}}{=} & \bigwedge_{i=1}^{m} \left(\bigvee_{x_{j,k} \in C_i} \overline{x_{j,k}} \right) \wedge \bigwedge_{i=1}^{m} \left(\bigwedge_{x_{j,k} \in C_i} (y_i \vee \overline{x_{j,k}}) \right) \\
& \wedge (\overline{y_1} \vee \overline{y_2} \vee z_1) \wedge \bigwedge_{\ell=1}^{m-3} (\overline{z_\ell} \vee \overline{y_{\ell+2}} \vee z_{\ell+1}) \wedge (\overline{z_{m-2}} \vee \overline{y_m} \vee q') \\
& \wedge \bigwedge_{i=1}^{n} ((\overline{x_{i,1}} \vee x_{i,2} \vee \overline{q_{i,1}})(\overline{x_{i,2}} \vee x_{i,3} \vee \overline{q_{i,2}})(\overline{x_{i,3}} \vee x_{i,4} \vee \overline{q_{i,3}})(\overline{x_{i,4}} \vee x_{i,1} \vee \overline{q_{i,4}})) \\
& \wedge (q_0 \vee \overline{q_{1,1}}) \wedge \bigwedge_{i=1}^{n} ((q_{i,1} \vee \overline{q_{i,2}})\,(q_{i,2} \vee \overline{q_{i,3}})\,(q_{i,3} \vee \overline{q_{i,4}})\,(q_{i,4} \vee \overline{q_{i+1,1}})) \\
& \wedge (q_{n+1,1} \vee \overline{q_0} \vee \overline{q'}).
\end{aligned}
$$

Each clause in $\bigwedge_{i=1}^{m}(\bigvee_{x_{j,k} \in C_i} \overline{x_{j,k}})$ has three negative literals since each clause $C_i \in C_\varphi$ has three positive literals. Other clauses in φ_ϕ have at most one positive literal and at most two negative literals. Hence, φ_ϕ is a 3-Horn formula. The number of occurrences of each variable $x_{j,k} \in X_\varphi$ and $y_i \in Y$ is exactly four, and each variable $q_{j,k} \in Q \setminus \{q_0, q', q_{n+1,1}\}$ appears exactly three times. Also, each variable $z_i \in Z$ and $q_0, q', q_{n+1,1}$ appears exactly twice. Therefore, φ_ϕ is an instance of DisConn 3-Horn-4.

We show the following claim about the satisfying assignment of φ_ϕ.

▷ Claim 21. Let α be an arbitrary NAE-satisfying assignment of ϕ, and let α' be a satisfying assignment of φ_ϕ such that $\alpha(x_{i,1}) = \alpha(x_{i,2}) = \alpha(x_{i,3}) = \alpha(x_{i,4}) = \alpha_i$ for all i $(1 \le i \le n)$, then α' assigns

(a) the value 1 to all the variables $y_i \in Y$,

(b) the value 1 to all the variables $z_i \in Z$ and the variable q',

(c) the same value to all the variables $q_{i,j} \in Q$ and the variable q_0.

Proof. From $\bigwedge_{i=1}^{m}(\bigwedge_{x_{j,k} \in C_i}(y_i \vee \overline{x_{j,k}}))$ and α assigns 1 to at least one variable in each $C_i \in C_\phi$, α' must assign 1 to all the variables $y_i \in Y$. In this case, the second line of φ_ϕ forces all the values of $z_i \in Z$ and q' to be assigned 1 for satisfying φ_ϕ. Furthermore, $q' = 1$ forces the values of all variables $q_0, q_{i,j} \in Q$ to be the same from the last two lines of φ_ϕ. ◁

For simplicity of proving the correctness of our reduction, we further transform φ_ϕ to Φ_{φ_ϕ} as follows, recalling Equation (1):

$$\Phi_{\varphi_\phi} = \varphi_\phi \wedge \bigwedge_{i=1}^{m} \left(\overline{y_i} \vee \bigvee_{x_{j,k} \in C_i} x_{j,k} \right)$$
$$\wedge (\overline{z_1} \vee y_1 y_2) \wedge \bigwedge_{\ell=1}^{m-3} (\overline{z_{\ell+1}} \vee y_{\ell+2} z_\ell) \wedge (\overline{q'} \vee y_m z_{m-2})$$
$$\wedge \bigwedge_{i=1}^{n} ((\overline{x_{i,2}} \vee x_{i,1} q_{i,1})(\overline{x_{i,3}} \vee x_{i,2} q_{i,2})(\overline{x_{i,4}} \vee x_{i,3} q_{i,3})(\overline{x_{i,1}} \vee x_{i,4} q_{i,4}))$$
$$\wedge (\overline{q_0} \vee q_{1,1}) \wedge \bigwedge_{i=1}^{n} ((\overline{q_{i,1}} \vee q_{i,2})\,(\overline{q_{i,2}} \vee q_{i,3})\,(\overline{q_{i,3}} \vee q_{i,4})\,(\overline{q_{i,4}} \vee q_{i+1,1}))$$
$$\wedge (\overline{q_{n+1,1}} \vee q_0 q').$$

We need two auxiliary lemmas to prove the NP-hardness of DisConn 3-Horn-4.

▶ **Lemma 22.** *Φ_{φ_ϕ} has a non-zero satisfying assignment if ϕ has an NAE-satisfying assignment.*

Proof. Let α be an NAE-satisfying assignment of ϕ, and let us construct a satisfying assignment α' of Φ_{φ_ϕ} from α. For each i ($1 \le i \le n$), we set the value of each variable $x_{i,k}$ ($1 \le k \le 4$) in α' to the value of x_i in α. Since α is an NAE-satisfying assignment of monotone formulas, there exists at least one variable x_i assigned 1. Thus, $x_{i,1}, x_{i,2}, x_{i,3}$, and $x_{i,4}$ are also assigned 1. This implies $q_{i,1}, q_{i,2}, q_{i,3}$, and $q_{i,4}$ must be 1 from the third line of Φ_{φ_ϕ}. From Claim 21, all the variables except for $x_{i,k}$ must be assigned 1 in α'. Clearly, α' is a non-zero assignment and satisfies Φ_{φ_ϕ}. ◀

▶ **Lemma 23.** *ϕ has an NAE-satisfying assignment if Φ_{φ_ϕ} has a non-zero satisfying assignment.*

Proof. Let α be an arbitrary non-zero satisfying assignment of Φ_{φ_ϕ}. We consider two cases for assigning to the variable q_0.

(a) In the case $\alpha(q_0) = 0$, we show that Φ_{φ_ϕ} cannot have any non-zero satisfying assignments. We consider the formula Φ'_{φ_ϕ} by removing the formula φ_ϕ from Φ_{φ_ϕ}. From the clauses $(\overline{q_0} \vee q_{1,1}) \wedge \bigwedge_{i=1}^{n}((\overline{q_{i,1}} \vee q_{i,2})(\overline{q_{i,2}} \vee q_{i,3})(\overline{q_{i,3}} \vee q_{i,4})(\overline{q_{i,4}} \vee q_{i+1,1})) \wedge (\overline{q_{n+1,1}} \vee q_0 q')$ in Φ'_{φ_ϕ} and $\alpha(q_0) = 0$, all variable $q_{i,j} \in Q$ must be assigned 0 under α. Then, the formula Φ'_{φ_ϕ} is simplified as follows.

$$\bigwedge_{i=1}^{m} \left(\overline{y_i} \vee \bigvee_{x_{j,k} \in C_i} x_{j,k} \right) \wedge (\overline{z_1} \vee y_1 y_2) \wedge \bigwedge_{\ell=1}^{m-3} (\overline{z_{\ell+1}} \vee y_{\ell+2} z_\ell) \wedge (\overline{q'} \vee y_m z_{m-2})$$
$$\wedge \bigwedge_{i=1}^{n} (\overline{x_{i,2}} \wedge \overline{x_{i,3}} \wedge \overline{x_{i,4}} \wedge \overline{x_{i,1}})$$

To satisfy this formula, α assigns

(1) the value 0 to all variables $x_{i,k} \in X_\varphi$ from $\bigwedge_{i=1}^{n} (\overline{x_{i,2}} \wedge \overline{x_{i,3}} \wedge \overline{x_{i,4}} \wedge \overline{x_{i,1}})$,

(2) the value 0 to all variables $y_i \in Y$ from $\bigwedge_{i=1}^{m}(\overline{y_i} \vee \bigvee_{x_{j,k} \in C_i} x_{j,k})$,

(3) the value 0 to all variables $z_i \in Z$ from $(\overline{z_1} \vee y_1 y_2) \wedge \bigwedge_{\ell=1}^{m-3}(\overline{z_{\ell+1}} \vee y_{\ell+2} z_\ell)$,

(4) the value 0 to q' from $(\overline{q'} \vee y_m z_{m-2})$

Hence, α must be the all-zero satisfying assignment. Therefore, Φ_{φ_ϕ} cannot have any non-zero satisfying assignment.

(b) In the case $\alpha(q_0) = 1$, from the clauses $(\overline{q_0} \vee q_{1,1}) \wedge \bigwedge_{i=1}^{n}((\overline{q_{i,1}} \vee q_{i,2})(\overline{q_{i,2}} \vee q_{i,3})(\overline{q_{i,3}} \vee q_{i,4})(\overline{q_{i,4}} \vee q_{i+1,1})) \wedge (\overline{q_{n+1,1}} \vee q_0 q')$ in Φ_{φ_ϕ} and $\alpha(q_0) = 1$, all variables $q_{i,j}$ and $q' \in Q$ must be assigned 1 under α. Then, $(\overline{z_1} \vee y_1 y_2) \wedge \bigwedge_{\ell=1}^{m-3}(\overline{z_{\ell+1}} \vee y_{\ell+2} z_\ell) \wedge (\overline{q'} \vee y_m z_{m-2})$ is simplified to $(\overline{z_1} \vee y_1 y_2) \wedge \bigwedge_{\ell=1}^{m-3}(\overline{z_{\ell+1}} \vee y_{\ell+2} z_\ell) \wedge (y_m z_{m-2})$. To satisfy these clauses, the values of all variables $y_i \in Y$ and $z_i \in Z$ are forced to 1 under α. Then, the formula Φ_{φ_ϕ} is simplified as follows:

$$\bigwedge_{i=1}^{m}\left(\left(\bigvee_{x_{j,k}\in C_i}\overline{x_{j,k}}\right) \wedge \left(\bigvee_{x_{j,k}\in C_i} x_{j,k}\right)\right)$$
$$\wedge \bigwedge_{i=1}^{n}((\overline{x_{i,1}} \vee x_{i,2})(\overline{x_{i,2}} \vee x_{i,3})(\overline{x_{i,3}} \vee x_{i,4})(\overline{x_{i,4}} \vee x_{i,1}))$$
$$\wedge \bigwedge_{i=1}^{n}((x_{i,1} \vee \overline{x_{i,2}})(x_{i,2} \vee \overline{x_{i,3}})(x_{i,3} \vee \overline{x_{i,4}})(x_{i,4} \vee \overline{x_{i,1}}))\,.$$

Using $(x \vee \overline{y})(\overline{x} \vee y) = (x \leftarrow y)(x \rightarrow y) = (x \leftrightarrow y)$, then the above formula is

$$\bigwedge_{i=1}^{m}\left(\left(\bigvee_{x_{j,k}\in C_i}\overline{x_{j,k}}\right) \wedge \left(\bigvee_{x_{j,k}\in C_i} x_{j,k}\right)\right)$$
$$\wedge \bigwedge_{i=1}^{n}((x_{i,1} \leftrightarrow x_{i,2})(x_{i,2} \leftrightarrow x_{i,3})(x_{i,3} \leftrightarrow x_{i,4})(x_{i,4} \leftrightarrow x_{i,1}))\,.$$

To satisfy this formula, each variable $x_{i,1}, x_{i,2}, x_{i,3}, x_{i,4}$ must be assigned the same value from $\bigwedge_{i=1}^{n}((x_{i,1} \leftrightarrow x_{i,2})(x_{i,2} \leftrightarrow x_{i,3})(x_{i,3} \leftrightarrow x_{i,4})(x_{i,4} \leftrightarrow x_{i,1}))$. We now construct an assignment α' by setting α'_i to the value of $x_{i,1}$ in α. The first line of the above formula assures that the assignment α' assigns 0 to at least one variable and 1 to at least one variable in each clause $C_i \in C_\phi$. Thus, α' is an NAE-satisfying assignment of ϕ.

From the above argument, this lemma holds. ◀

We are ready to prove the following Theorem.

▶ **Theorem 24.** DisConn 3-Horn-4 *is* NP*-complete.*

Proof. Let ϕ be an instance of Monotone NAE E3-SAT-E4. Let φ_ϕ be a 3-Horn formula constructed from ϕ, and Φ_{φ_ϕ} be the formula constructed from φ_ϕ. From Lemma 10, if Φ_{φ_ϕ} has a non-zero satisfying assignment, the solution graph G_{φ_ϕ} is disconnected. Thus, DisConn 3-Horn-4 belongs to NP. Lemma 22 and Lemma 23 imply that DisConn 3-Horn-4 is NP-hard. Therefore, DisConn 3-Horn-4 is NP-complete. ◀

The coNP-completeness of Conn 3-Horn-4 immediately follows from Theorem 24. We then show the following Theorem.

▶ **Theorem 4.** Conn 3-Horn *is* coNP*-complete even when each variable appears exactly four times.*

Proof. We prove the coNP-hardness of Conn 3-Horn-E4 by a polynomial-time reduction from Conn 3-Horn-4. We first show the construction of a 3-Horn formula with each variable appearing exactly four times from an instance of Conn 3-Horn-4. Let ϕ be an instance of Conn 3-Horn-4. Then, for every variable v which appears three times, we introduce new variables v_1, v_2 and add four clauses $(\overline{v_1} \vee \overline{v_2})(\overline{v_1} \vee \overline{v_2})(\overline{v_1} \vee \overline{v_2})(\overline{v_1} \vee \overline{v_2} \vee \overline{v})$. For every

variable that appears at most twice, we repeat the same operation until each variable appears four times. Thus, we obtain an instance φ of Conn 3-Horn-E4. Then, we transform ϕ and φ to Φ_ϕ and Φ_φ, respectively, for simplicity of proving the correctness of our reduction. From the above construction, Φ_φ has unit clauses with negative literals of all added variables. Therefore, Φ_φ is equivalent to Φ_ϕ by assigning 0 to all added variables of φ. It means that Φ_ϕ has a non-zero satisfying assignment if and only if Φ_φ has a non-zero satisfying assignment, and by Lemma 10 this completes the proof. ◀

By duality, coNP-completeness of the Boolean connectivity for dual-Horn formulas with the same restrictions follows from the same discussion.

5 Conclusion

We proved that Conn 3-Horn remains coNP-complete even when the length of each clause is at most three and each variable appears exactly four times. Furthermore, we provided a polynomial-time algorithm for Conn 3-Horn with each variable appearing at most three times and the length of each clause exactly three. These results seem to give a borderline between P and coNP-complete. However, it remains the complexity of two cases of Conn 3-Horn:(a) the length of each clause is exactly three and each variable appears exactly or at most four times; (b) the length of each clause is at most three and each variable appears exactly or at most three times. The complexity of Monotone NAE 3-SAT might be closely related to that of Conn 3-Horn for the following reasons. Monotone NAE 3-SAT is polynomial-time solvable when the length of each clause is exactly three and each variable appears at most three times in [11]. Monotone NAE 3-SAT remains NP-complete even when the length of each clause is at most three and each variable appears exactly four times. These complexities are equivalent to those of DisConn 3-Horn with the same restrictions. Furthermore, Monotone NAE 3-SAT is NP-complete even when the length of each clause is at most three and each variable appears exactly three times [4]. It also remains NP-complete even when the length of each clause is exactly three and each variable appears exactly four times [3]. Thus, we conjecture that cases (a) and (b) are coNP-complete.

References

1 Paul S. Bonsma, Amer E. Mouawad, Naomi Nishimura, and Venkatesh Raman. The Complexity of Bounded Length Graph Recoloring and CSP Reconfiguration. In *Proceedings of the 9th International Symposium on Parameterized and Exact Computation (IPEC 2014)*, volume 8894 of *Lecture Notes in Computer Science*, pages 110–121. Springer, 2014. doi:10.1007/978-3-319-13524-3_10.

2 Jean Cardinal, Erik D. Demaine, David Eppstein, Robert A. Hearn, and Andrew Winslow. Reconfiguration of Satisfying Assignments and Subset Sums: Easy to Find, Hard to Connect. In *Proceedings of 24th International Computing and Combinatorics Conference (COCOON 2018)*, volume 10976 of *Lecture Notes in Computer Science*, pages 365–377. Springer, 2018. doi:10.1007/978-3-319-94776-1_31.

3 Andreas Darmann and Janosch Döcker. On a simple hard variant of Not-All-Equal 3-Sat. *Theoretical Compututer Science*, 815:147–152, 2020. doi:10.1016/J.TCS.2020.02.010.

4 Ali Dehghan, Mohammad-Reza Sadeghi, and Arash Ahadi. On the Complexity of Deciding Whether the Regular Number is at Most Two. *Graphs and Combinatorics*, 31(5):1359–1365, 2015. doi:10.1007/S00373-014-1446-9.

5 Oya Ekin, Peter L. Hammer, and Alexander Kogan. On Connected Boolean Functions. *Discrete Applied Mathematics*, 96-97:337–362, 1999. doi:10.1016/S0166-218X(99)00098-0.

6 Parikshit Gopalan, Phokion G. Kolaitis, Elitza N. Maneva, and Christos H. Papadimitriou. The Connectivity of Boolean Satisfiability: Computational and Structural Dichotomies. *SIAM Journal on Computing*, 38(6):2330–2355, 2009. doi:10.1137/07070440X.

7 Kazuhisa Makino, Suguru Tamaki, and Masaki Yamamoto. On the Boolean connectivity problem for Horn relations. *Discrete Applied Mathematics*, 158(18):2024–2030, 2010. doi:10.1016/J.DAM.2010.08.019.

8 Kazuhisa Makino, Suguru Tamaki, and Masaki Yamamoto. An exact algorithm for the Boolean connectivity problem for k-CNF. *Theoretical Computer Science*, 412(35):4613–4618, 2011. doi:10.1016/j.tcs.2011.04.041.

9 Amer E. Mouawad, Naomi Nishimura, Vinayak Pathak, and Venkatesh Raman. Shortest Reconfiguration Paths in the Solution Space of Boolean Formulas. In *Proceedings of the 42nd EATCS International Colloquium on Automata, Languages, and Programming (ICALP 2015)*, volume 9134 of *Lecture Notes in Computer Science*, pages 985–996. Springer, 2015. doi:10.1007/978-3-662-47672-7_80.

10 Ramamohan Paturi, Pavel Pudlák, and Francis Zane. Satisfiability Coding Lemma. *Chicago Journal of Theoretical Computer Science*, 1999. URL: http://cjtcs.cs.uchicago.edu/articles/1999/11/contents.html.

11 Stefan Porschen, Bert Randerath, and Ewald Speckenmeyer. Linear Time Algorithms for Some Not-All-Equal Satisfiability Problems. In *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT 2003)*, volume 2919 of *Lecture Notes in Computer Science*, pages 172–187. Springer, 2003. doi:10.1007/978-3-540-24605-3_14.

12 Thomas J. Schaefer. The Complexity of Satisfiability Problems. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing (STOC 1978)*, pages 216–226. ACM, 1978. doi:10.1145/800133.804350.

13 Patrick Scharpfenecker. On the Structure of Solution-Graphs for Boolean Formulas. In *Proceedings of the 20th International Symposium (FCT 2015)*, volume 9210 of *Lecture Notes in Computer Science*, pages 118–130. Springer, 2015. doi:10.1007/978-3-319-22177-9_10.

14 Patrick Scharpfenecker and Jacobo Torán. Solution-Graphs of Boolean Formulas and Isomorphism. *Journal on Satisfiability, Boolean Modelling and Computation*, 10(1):37–58, 2016. doi:10.3233/SAT190113.

15 Konrad W. Schwerdtfeger. A Computational Trichotomy for Connectivity of Boolean Satisfiability. *Journal on Satisfiability, Boolean Modelling and Computation*, 8(3/4):173–195, 2014. doi:10.3233/sat190097.

Complexity of Local Search for CSPs Parameterized by Constraint Difference

Aditya Anand
University of Michigan, Ann Arbor, MI, USA

Vincent Cohen-Addad
Google Research, NYC, USA

Tommaso D'Orsi
Bocconi University, Milan, Italy

Anupam Gupta
New York University, NY, USA

Euiwoong Lee
University of Michigan, Ann Arbor, MI, USA

Debmalya Panigrahi
Duke University, Durham, NC, USA

Sijin Peng
Massachusetts Institute of Technology, Cambridge, MA, USA

Abstract

In this paper, we study the parameterized complexity of local search, whose goal is to find a good nearby solution from the given current solution. Formally, given an optimization problem where the goal is to find the largest *feasible* subset S of a universe U, the new input consists of a *current solution* P (not necessarily feasible) as well as an ordinary input for the problem. Given the existence of a feasible solution S^*, the goal is to find a feasible solution as good as S^* in parameterized time $f(k) \cdot n^{O(1)}$, where k denotes the *distance* $|P \Delta S^*|$. This model generalizes numerous classical parameterized optimization problems whose parameter k is the minimum number of elements removed from U to make it feasible, which corresponds to the case $P = U$.

We apply this model to widely studied Constraint Satisfaction Problems (CSPs), where U is the set of constraints, and a subset U' of constraints is feasible if there is an assignment to the variables satisfying all constraints in U'. We give a complete characterization of the parameterized complexity of all boolean-alphabet *symmetric* CSPs, where the predicate's acceptance depends on the number of true literals.

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

Keywords and phrases Constraint Satisfaction Problems, Parameterized Local Search, Optimization

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.26

Related Version *Full Version*: https://arxiv.org/abs/2512.03275

Funding *Anupam Gupta*: Supported in part by NSF awards CCF-2224718 and CCF-2422926.
Euiwoong Lee: Supported in part by NSF award CCF-2236669 and by Google, Inc.
Debmalya Panigrahi: Supported in part by NSF awards CCF-1955703 and CCF-2329230.

1 Introduction

The classical theory of NP-hardness sets limits on the polynomial-time tractability of a vast array of algorithmic problems. This raises the question: *what additional information about the problem instance can one use to help overcome complexity barriers?* A very successful line of research that aims to address this broad question is that of *fixed parameter tractable* (FPT) algorithms. The idea is to identify a parameter that has a small value in *typical* instances, and design an algorithm whose running time is exponential in the value of the parameter but polynomial in the overall size of the instance.

This philosophy has been widely applied to the following general class of optimization problems: let U be a universe and let $\mathcal{S} \subseteq 2^U$ be an (often implicitly given) collection of *feasible* subsets of U, and the goal is to find $S \in \mathcal{S}$ that maximizes $|S|$, or equivalently to find

© Aditya Anand, Vincent Cohen-Addad, Tommaso D'Orsi, Anupam Gupta, Euiwoong Lee, Debmalya Panigrahi, and Sijin Peng;
licensed under Creative Commons License CC-BY 4.0

the smallest T that puts $U \setminus T \in \mathcal{S}$. For instance, given a graph $G = (V, E)$, if $U = E$ and $F \in \mathcal{S}$ if (V, F) is 2-colorable, the problem corresponds to the famous MaxCut/MinUncut pair. Or given $G = (V, E)$, if $U = V$ and $\mathcal{S}$ denotes the collection of all independent sets, then we have Independent Set/Vertex Cover. In this class, there are many situations where typical (or at least interesting) instances have the optimal value for the minimization version much smaller than $|U|$. (I.e., the universe U is already close to feasible.) Consequently, numerous results in the FPT literature study a problem from the above class - we let the parameter k be the optimal value for the minimization version, and give an algorithm of running time $f(k)n^{O(1)}$ (better than the naive running time of $n^{O(k)}$).

Can we extend the applicability of this framework? We can interpret the algorithms in the above paragraph as one-step *local search*, where we are given a current solution (not necessarily feasible) *close* to some feasible solution S^* with the optimal value, and the algorithm finds a feasible solution as good as S^*. This motivates us to ask: *if the algorithm is provided* **any** *current solution $P \subseteq U$ (as a solution for the max version) and there is a feasible solution $S^* \subseteq U$ close to P, can the algorithm efficiently recover a solution as good as S^*?* Formally, we define the parameter k as the distance $|P \Delta S^*|$, and our goal is to design an algorithm whose runtime is arbitrary in k but polynomial in the size of the problem instance.[1]

We apply this model to Constraint Satisfaction Problems (CSPs), one of the most important classes of optimization problems in the theory of computation. Given a predicate $R \subseteq \{0,1\}^r$, MaxCSP(R) is the computational problem whose input consists of the set of variables V and the set of constraints $\mathcal{C}$. Given an assignment $\alpha : V \to \{0,1\}$, each constraint $C \in \mathcal{C}$ is satisfied when the values of the r specified literals (variables or their negations) belong to R. (See Section 2 for formal definitions.) The goal is to find an assignment that satisfies as many constraints as possible. Generalizing the MaxCut example above, we apply MaxCSP(R) to our model where $U = \mathcal{C}$ and a subset of constraints $\mathcal{C}' \subseteq \mathcal{C}$ is feasible if there exists an assignment α that satisfies all of $\mathcal{C}'$ (i.e., $\mathcal{C}'$ is *satisfiable*); so the current solution $\mathcal{P}$ is a set of constraints and we assume that it is close to some set of satisfiable constraints. It yields the following computational task.

Problem: ImproveMaxCSP(R)
Input: A CSP(R) instance $I(V, \mathcal{C})$, an integer $k \in \mathbb{N}$, and $\mathcal{P} \subseteq \mathcal{C}$.
Parameter: k
Promise: There exists an assignment α_0 with the property $|\mathcal{C}_I(\alpha_0) \Delta \mathcal{P}| \leq k$, where $\mathcal{C}_I(\alpha_0)$ is the set of clauses α_0 satisfies, and Δ represents symmetric difference.
Output: A *good* assignment α, where the word *good* simply means $|\mathcal{C}_I(\alpha)| \geq |\mathcal{C}_I(\alpha_0)|$. (We do not require α to satisfy $|\mathcal{C}_I(\alpha) \Delta \mathcal{P}| \leq k$.)

Note that, given an instance $(I(V, \mathcal{C}), k, \mathcal{P})$, the problem is essentially equivalent to find α such that $|\mathcal{C}_I(\alpha)|$ is at least $\max_{\beta : |\mathcal{C}_I(\beta) \Delta \mathcal{P}| \leq k} |\mathcal{C}_I(\beta)|$; in words, find a feasible solution as good as any feasible k-neighbor of $\mathcal{P}$. (Formally, we still will keep the fixed promised assignment α_0 as part of the instance and define a good assignment based on α_0.)

We also observe that if the decision version of CSP(R) (i.e., finding an assignment that satisfies every constraint) can be solved in polynomial time, an easy $|\mathcal{C}|^k \cdot \mathrm{poly}(|V|, |\mathcal{C}|)$-time algorithm for ImproveMaxCSP(R) follows by exhaustively guessing $(\mathcal{C}_I(\alpha_0) \Delta \mathcal{P})$ and

[1] Actually, since $|S \Delta S^*| = |(U \setminus S) \Delta (U \setminus S^*)|$, this parameter remains the same for both maximization and minimization versions.

solving the decision version with constraints $\mathcal{C}_I(\alpha_0)$; otherwise it is NP-hard even when $k = 0$ (just letting $\mathcal{P} = \mathcal{C}$), which concludes that ImproveMaxCSP(R) is in **XP** if and only if the decision version is in **P**. Also, if MaxCSP(R) is NP-hard, there is no poly($|V|, |\mathcal{C}|, k$)-time algorithm for ImproveMaxCSP(R).

In this paper, we provide a complete characterization of the parameterized complexity of ImproveMaxCSP(R), where R is restricted to a *symmetric* predicate, whose acceptance only depends on the number of true literals. This still includes many well-known CSPs including rLIN (which generalizes MaxCut), rSAT, rAND, rNAE, and rAE, where AE and NAE abbreviate All-Equal and Not-All-Equal respectively. Namely, the main contribution of this paper is the following theorem:

▶ **Theorem 1** (Main Theorem (Informal)). *For any symmetric predicate R, ImproveMax CSP(R) is either FPT or W[1]-hard.*

It turns out that among nontrivial predicates, rAND for any $r \geq 1$ and 2AE (generalizing MaxCut) are the only ones in FPT. We design parameterized algorithms for them extending the previous algorithms for MinCSP. While many predicates inherit their hardness result from the hardness of MinCSP, we also prove hardness for the problems whose MinCSP is tractable, namely rAE for $r \geq 3$ and ≤ 1-out-of-rSAT for $r \geq 2$. See Section 2.3 for our formal characterization.

In the context of local search, a natural special case of ImproveMaxCSP(R) is when the input $\mathcal{P}$ is restricted to be $\mathcal{C}_I(\beta)$ for some assignment $\beta : V \to \{0, 1\}$. Theorem 1 addresses this special case as well, in the sense that our W[1]-hard reductions indeed have $\mathcal{P} = \mathcal{C}_I(\beta)$ for some assignment β.

Related Work and Discussion

The parameterized complexity of local search, whose goal is to improve the current solution to a strictly better solution nearby, was indeed studied in the parameterized complexity literature. Fellows, Fomin, Lokshtanov, Rosamond, Saurabh, and Villanger [8] study various problems like r-Center, Vertex Cover, Odd Cycle Transversal, Max-Cut, and Max-Bisection on special classes of sparse graphs like planar, minor-free, and bounded-degree graphs, and Szeider [25] considers a similar framework for SAT and MaxSAT.

This (strict) local search model can be relaxed to another model called *permissive* local search [19]. Here we are given a solution S, and the goal is to find a solution S' which has lower cost (which may differ from S in more than k vertices), or correctly conclude that every solution which differs from S in at most k vertices has higher cost. This model has been studied for a few problems - including a variant of stable marriage [19], Vertex Cover [9] and also in the context of Min Ones CSPs [14], extending Marx's complete classification of the parameterized complexity of CSPs based on if there is a solution with exactly k ones [17].

One crucial difference between our model and all previous models on CSPs is how the distance between the two solutions is defined. The previous papers define it as the number of variables with different values in two solutions, but our definition concerns the set of constraints whose satisfaction changes between two solutions. In fact, recall that our definition of a solution is an arbitrary set $\mathcal{P}$ of constraints not necessarily feasible (i.e., corresponding to a variable assignment), which allows $\mathcal{P} = \mathcal{C}$ and captures algorithms parameterized by the number of unsatisfied constraints. In that sense, our model is slightly broader than traditional local search where one maintains a feasible solution and tries to strictly improve in every step.

Fixed parameter tractability of CSPs parameterized by the number of unsatisfied constraints has been an important and active research area [2, 11, 12, 13], where a complete classification for Boolean relations was obtained very recently. Their definition of CSPs is strictly more general than ours, where each constraint language contains multiple predicates, even with different arities. It is an exciting research direction to extend our framework for every CSP.

Our negative results can be interpreted as demonstrating the importance of *near-perfect instances* for classical FPT algorithms. For instance, we show ImproveMaxCSP(2SAT) is $W[1]$-hard while MinCSP(2SAT) is known to admit an FPT algorithm [23, 22, 6, 21], so the fact that the optimal solution indeed satisfies all but k constraints affects the parameterized complexity.

There are several natural research directions based on this paper. We believe that the following directions will be interesting to study further:

1. The most immediate one is to extend our characterization for all CSPs. One notable set of boolean relations whose complexity is still open in our model is $\{x = y, x \wedge \neg y\}$.
2. A natural variant of our model is when the solution is represented as an assignment to the variables, which is guaranteed to have a Hamming distance at most k to a strictly better solution. For Min Ones CSP, where the goal is to find a satisfying assignment with the minimum number of true variables, this variant has been studied previously [14].
3. Other than CSPs, one framework that captures numerous results in the FPT literature is covering (resp. packing) problems, especially on graphs, where we want to select the smallest (resp. largest) subset $S \subseteq U$ subject to some covering (resp. packing) constraints. Local search naturally extends to this setting (i.e., we are given $S' \subseteq U$ with $|S' \Delta S| \leq k$), and it will be interesting to see if classical FPT results on graph covering/packing problems extend to this model, including Multicut [3, 18], Min Bisection [5], k-Cut [10], Disjoint Paths [24].
4. In the approximation algorithms literature, there are numerous studies about CSPs on *structured instances*, most notably dense and expanding instances (see [1] and references therein). It would be interesting to see whether these structures, if suitably parameterized, help the problems become more tractable in our model.

2 Our Results

2.1 Constraint Satisfaction Problems

In this section we recall basic definitions of constraint satisfaction problems.

For integer $r \in \mathbb{N}^+$, a *boolean relation* R is a subset of $\mathbb{F}_2^r$. The integer r is called the *arity* of R. For a boolean relation R of arity r and $b \in \mathbb{F}_2^r$, define the boolean relation $R \oplus b := \{\alpha \oplus b \mid \alpha \in R\}$, where $\oplus$ is the addition operator over $\mathbb{F}_2^r$. When considering R as a clause in a CSP instance, $\oplus$ operator can be seen as variable negations. For a relation R with index $[r]$ and $S \subseteq [r]$, define the projection relation $\pi_S(R) = \{\alpha_S \mid \alpha \in R\}$, where α_S means extracting the value on coordinates in S.

A boolean relation R of arity r is *symmetric* if for all $(x_1, \cdots, x_r) \in \mathbb{F}_2^r$ and any permutation $p : [r] \to [r]$, $(x_1, \cdots, x_r) \in R \iff (x_{p(1)}, \cdots, x_{p(r)}) \in R$. Equivalently, R is symmetric if there exists $S \subseteq \{0, 1, \cdots, r\}$ such that $(x_1, \cdots, x_r) \in R \iff \sum_{i=1}^{r} x_i \in S$ (where the addition is performed in $\mathbb{Z}$). We define SymRel(r, S) to be such R.

A *boolean constraint language* is a set of boolean relations. Two constraint languages are *equivalent* if they contain exactly the same set of relations. A *clause* C over a boolean constraint language Γ is a pair (V_C, R_C), where $R_C \in \Gamma$ and $V_C = (v_{C,1}, \ldots, v_{C,r})$ is an

r-tuple of variables where r is the arity of R_C. We refer to V_C as the *scope* of the clause C. An assignment $\alpha : V_C \to \{0,1\}$ *satisfies* clause (V_C, R_C) if $(\alpha(v_{C,1}), \ldots, \alpha(v_{C,r})) \in R_C$, and α *violates* the clause otherwise.

In this paper, we only consider a restricted set of boolean constraint languages, where the language Γ includes all possible negation patterns $R \oplus b$ for some symmetric predicate R of arity r and $b \in \mathbb{F}_2^r$. We use the notation SymLang-N$(R) = \{R \oplus b \mid b \in \mathbb{F}_2^r\}$ for such Γ, where SymLang-N stands for "symmetric language with negation". We further define SymLang-N(r, S) to be SymLang-N(SymRel(r, S)).

For a boolean constraint language Γ, a CSP(Γ) instance I is a pair $(V_I, \mathcal{C}_I)$, where V_I is the set of variables, and $\mathcal{C}_I$ is the set of clauses over the boolean constraint language Γ, all of which have scope inside V_I. We say a variable v is incident to a clause C if v belongs to the scope of C.

For an assignment $\alpha : V_I \to \{0,1\}$, define $\mathcal{C}_I(\alpha)$ to be the set of clauses α satisfies in I. We further define $cost_I(\alpha) = |\mathcal{C} \backslash \mathcal{C}_I(\alpha)|$, the number of unsatisfied clauses, to be the *cost* of the assignment. We define the *size* of an instance I as $\max(|V_I|, |\mathcal{C}_I|)$ and denote it by n_I. We ignore the subscripts when the instance is clear in the context.

Now we give formal definitions for the problems we consider. We first define MinCSP(Γ), and then define our central problem ImproveMaxCSP(Γ). In this paper, our main goal is to characterize the parameterized complexity of ImproveMaxCSP(Γ) when $\Gamma =$ SymLang-N(r, S) for some $r \in \mathbb{N}^+$ and $S \subseteq \{0, 1, \cdots, r\}$.

Problem: MinCSP(Γ)
Input: A CSP(Γ) instance $I(V, \mathcal{C})$, an integer $k \in \mathbb{N}$.
Parameter: k.
Output: An assignment α with $cost(\alpha) \leq k$ if it exists, otherwise report that such an assignment does not exist.

Problem: ImproveMaxCSP(Γ)
Input: A CSP(Γ) instance $I(V, \mathcal{C})$, an integer $k \in \mathbb{N}$, and $\mathcal{P} \subseteq \mathcal{C}$
Parameter: k
Promise: There exists an assignment $\alpha_0 : V_I \to \{0,1\}$ such that $|\mathcal{C}_I(\alpha_0) \Delta \mathcal{P}| \leq k$.
Output: A *good* assignment α, where the word *good* simply means $|\mathcal{C}_I(\alpha)| \geq |\mathcal{C}_I(\alpha_0)|$. (We do not require the output α to satisfy $|\mathcal{C}_I(\alpha) \Delta \mathcal{P}| \leq k$.)

From the definition, MinCSP(Γ) is a special case of ImproveMaxCSP(Γ) by setting $\mathcal{P}$ to be the constraint set $\mathcal{C}$.

A boolean constraint language Γ is *trivial* if $\Gamma = \varnothing$ or $\Gamma = \{\{0,1\}^r\}$ for some $r \in \mathbb{N}^+$ and *nontrivial* otherwise. When Γ is trivial, MinCSP(Γ) and ImproveMaxCSP(Γ) can be solved by trivial algorithms. So in the following we only consider nontrivial boolean constraint languages.

2.2 Abbreviations for Special CSPs

For simplicity, we abbreviate boolean constraint languages mentioned in the paper. Here we introduce the following simple but useful claim in identifying equivalence between boolean constraint languages.

▷ Claim 2. SymLang-N(r, S) and SymLang-N$(r, \{r - x \mid x \in S\})$ are equivalent boolean constraint languages.

Proof. From SymRel$(r, S) + (1, 1, \cdots, 1) =$ SymRel$(r, \{r - x \mid x \in S\})$, where $(1, 1, \cdots, 1)$ is the element in $\mathbb{F}_2^r$ that has value 1 in each coordinate, we have SymRel$(r, S) + b =$ SymRel$(r, \{r - x \mid x \in S\}) + (b + (1, 1, \cdots, 1))$ for any $b \in \mathbb{F}_2^r$. ◁

We will use the following abbreviations in this paper.

- rAND = SymLang-N$(r, \{0\})$ = SymLang-N$(r, \{r\})$.
- rSAT = SymLang-N$(r, \{1, 2, \cdots, r\})$ = SymLang-N$(r, \{0, 1, \cdots, r-1\})$.
- rAE (All Equal) = SymLang-N$(r, \{0, r\})$.
- $\leq$ 1-out-of-rSAT = SymLang-N$(r, \{0, 1\})$ = SymLang-N$(r, \{r-1, r\})$.

We also use abbreviations for the corresponding MinCSP and ImproveMaxCSP problems. For example, ImproveMaxCSP(rAND) corresponds to the problem ImproveMaxCSP(SymLang-N$(r, \{0\})$).

2.3 Our Characterization

Having defined our framework, we first provide the formal version of Theorem 1:

▶ **Theorem 3** (Main Theorem (Formal))**.** *For any $r \in \mathbb{N}^+$ and $S \subseteq \{0, 1, \cdots, r\}$, the problem* ImproveMaxCSP(SymLang-N(r, S)) *is either FPT or W[1]-hard.*

As MinCSP(Γ) is a special case of ImproveMaxCSP(Γ), the W[1]-hardness of the former implies that of the latter. The recent characterization of parameterized tractability of MinCSP(Γ) [13] implies the following theorem.

▶ **Theorem 4.** *For nontrivial $\Gamma =$ SymLang-N(r, S) for some $r \in \mathbb{N}^+$ and $S \subseteq \{0, 1, \cdots, r\}$ that is not equivalent to rAND, rAE or $\leq$ 1-out-of-rSAT, ImproveMaxCSP(Γ) is W[1]-hard.*

It remains to study the remaining problems.

▶ Remark 5. We say that an algorithm A *solves* ImproveMaxCSP(Γ), if for any instance that satisfies the promise, A outputs a good assignment. However, the run time is measured on all possible instances. In other words, A runs in $f(k)n^{O(1)}$ time if it terminates and outputs some assignment in $f(k)n^{O(1)}$ time no matter whether the input instance $(I, k, \mathcal{P})$ satisfies the promise. Clearly, only the run time on instances satisfying the promise is critical, as we can always set a run time limit to the algorithm to rule out instances that require too much time to indicate their violation of promise.

For ImproveMaxCSP(rAND), a color-coding combined with the LP rounding for Densest Subhypergraph yields the following result proved in Section 3.

▶ **Theorem 6.** *There exists an algorithm that solves ImproveMaxCSP(rAND) in $2^{O(k \log k)} n^{O(1)}$ time for all $r \geq 1$.*

For rAE, its tractability depends on r. When $r = 2$, we use the celebrated *unbreakability* framework [10, 4, 5, 16] to design a parameterized algorithm in Section 4.

▶ **Theorem 7.** *There exists an algorithm that solves ImproveMaxCSP(2AE) in time $2^{O(k^2)} n^{O(1)}$.*

However, for rAE with $r \geq 3$, we show nontrivial reductions from Paired Minimum st-Cut to show the following hardness.

▶ **Theorem 8.** *ImproveMaxCSP(rAE) is W[1]-hard for all $r \geq 3$.*

Finally, for $\leq$ 1-out-of-rSAT, we prove that even the first nontrivial case $r = 2$, which is equivalent to 2SAT, is already W[1]-hard. Even though MinCSP(2SAT) is in FPT, this result follows from the (slightly informal) fact that *Vertex Cover and Independent Set are equivalent in our model.*

▶ **Theorem 9.** *ImproveMaxCSP($\leq$ 1-out-of-rSAT) is W[1]-hard for any $r \geq 2$.*

The hardness proofs can be found in the full version. It is easy to see that the other theorems imply Theorem 3 in a straightforward way.

3 FPT Algorithm for rAND

The main goal of this section is to prove Theorem 6. During the course of the algorithm, we create instances involving AND constraints of different arities. Hence, we will show a slightly stronger result:

▶ **Theorem 10.** *There is an algorithm that solves ImproveMaxCSP($\cup_{r' \leq r} r'$AND) in $2^{O(k \log k)} n^{O(1)}$ time for all $r \geq 1$.*

Since $\cup_{r' \leq r} r'\text{AND} \supseteq r\text{AND}$, Theorem 10 immediately shows that for any $r \geq 1$, the problem ImproveMaxCSP(rAND) is FPT.

3.1 Preliminaries

In this section we recall basic definitions about hypergraphs. A *hypergraph* $H(V, E)$ is a generalization of graph where an edge can contain multiple vertices. For $e \in E$ and $v \in V$, say v is incident to e and e is incident to v if e contains v. For a hypergraph H and $V_0 \subseteq V$, we denote $H(V_0)$ as the *induced subhypergraph* of V_0 in H, and denote $E(H(V_0))$ to be the edge set of the induced subhypergraph.

In the main algorithm in this section (provided in Theorem 14), we need to solve the following problem:

Problem: Maximum Induced Subhypergraph with Vertex Weights.
Input: A hypergraph $H(V, E)$ and a weight function $w : V \to \mathbb{Z}$.
Output: A vertex subset $V_0 \subseteq V$ that maximizes $|E(H(V_0))| - \sum_{v \in V_0} w(v)$.

The problem is closely related to Densest Subgraph (See [15] for a survey) and has a polynomial-time algorithm based on similar algorithms for Densest Subgraph. One algorithm is to observe that $|E(H(V_0))| - \sum_{v \in V_0} w(v)$ is supermodular and run a submodular minimization algorithm.[2] The below is an LP-based algorithm specific to our setting.

▶ **Lemma 11.** *There exists a polynomial-time exact algorithm that solves Maximum Induced Subhypergraph with Vertex Weights.*

[2] We thank an anonymous reviewer for this observation.

Proof. Consider the following LP:

$$\begin{array}{lll} \text{maximize} & \sum_{e\in E} x_e - \sum_{v\in V} w(v)y_v & \\ \text{subject to} & x_e \le y_v & \forall e \in E, v \in V \text{ s.t. } e \text{ is incident to } v \\ & x_e \in [0,1] & \forall e \in E \\ & y_v \in [0,1] & \forall v \in V \end{array}$$

Since we use $x_e \le y_v$ to mimic the constraint that "one should select a vertex before selecting incident edges", if the LP is integral (i.e. $x_e, y_v \in \{0,1\}$) then the set of vertices v with $y_v = 1$ in the optimal solution of the LP corresponds to the correct output to the Maximum Induced Subhypergraph with Vertex Weights problem.

The algorithm solves the LP, providing optimal fractional solution $(x_e^\star, y_v^\star)$, and rounds the solution to an integer solution in the following way: Arbitrarily select $p \in (0,1]$ and set $x_e^p = [x_e^\star \ge p], y_v^p = [y_v^\star \ge p]$, where $[P]$ denotes the Iverson bracket and equals to 1 if P is true otherwise 0 for some statement P. The solution is clearly an integral solution to the LP.

Now we show that any such p produces an integral solution that has the same value as that of $(x_e^\star, y_v^\star)$, denoted as $f^\star$. Define $f(p)$ to be the objective function value of the solution (x_e^p, y_v^p). Clearly we have $f(p) \le f^\star$. We further have

$$\begin{aligned} \int_0^1 f(p)\mathrm{d}p &= \int_0^1 \left(\sum_{e\in E} [x_e^\star \ge p] - \sum_{v\in V} w(v)[y_v^\star \ge p] \right) \mathrm{d}p \\ &= \sum_{e\in E} x_e^\star - \sum_{v\in V} w(v) y_v^\star = f^\star \end{aligned} \tag{1}$$

So the set $\{x \in (0,1] \mid f(x) < f^\star\}$ has zero measure. Further from the fact that $f(p)$ is a step function of finite number of "steps" and each "step" has nonzero length, such set must be empty, and $f(p) = f^\star$. ◀

Our FPT algorithms in Section 3 and Section 4 will use the technique of *color coding* and its derandomization given below.

▶ **Theorem 12** ([20]). *Given a set U of size n and integers $0 \le a, b \le n$, one can in time $2^{O(\min(a,b)\log(a+b))} \cdot n \log n$ construct a family $\mathcal{F}$ of at most $2^{O(\min(a,b)\log(a+b))} \log n$ colorings $U \to \{0,1\}$ such that the following holds: for any sets $A, B \subseteq U$, $A \cap B = \varnothing$, $|A| \le a$, $|B| \le b$, there exists a coloring $\chi \in \mathcal{F}$ with $\chi(a) = 1$ for all $a \in A$ and $\chi(b) = 0$ for all $b \in B$.*

3.2 Algorithm with Variable Solution

Given an instance $(I, k, \mathcal{P})$, we say that $\mathcal{P}$ is *satisfiable* if some assignment satisfies all clauses in $\mathcal{P}$. We will give an algorithm that runs in $2^{O(k\log k)} n^{O(1)}$ time and solves ImproveMaxCSP($\cup_{r' \le r} r'$AND) when the instance satisfies an additional promise that $\mathcal{P}$ is satisfiable.

When this additional promise is satisfied, it is easy to find an assignment satisfying all the clauses of $\mathcal{P}$ in polynomial time, so in the following we assume that the instance $(I, k, \mathcal{P})$ is equipped with such an assignment α. In the full version, we show how to transform any instance into instances where this additional promise is true.

Equipped with this additional assignment, we then restructure the instance in the following way: Define $k' := k + |\mathcal{P} \Delta C_I(\alpha)|$ and $\mathcal{P}' := C_I(\alpha)$, and replace $(I, k, \mathcal{P}, \alpha)$ with $(I, k', \mathcal{P}', \alpha)$, to keep $\mathcal{P}$ to be maximal in accordance with α. In other words, we reset $\mathcal{P}$ to be the set of clauses satisfied by α. If the instance satisfies the promise (that some good assignment is

close to the $\mathcal{P}$), the good assignment satisfies at most $|\mathcal{P}| + k$ clauses, which implies that $|\mathcal{P}\Delta C_I(\alpha)| \leq k$, and $k' \leq 2k$. If $k' > 2k$, we return an arbitrary assignment (to handle cases where the promise is not met).

Now we show that, there exists an good assignment close to $\mathcal{P}$ whose Hamming distance to α is small.

▶ **Lemma 13.** *For a* ImproveMaxCSP($\cup_{r' \leq r} r'$AND) *instance* $(I, k, \mathcal{P}, \alpha)$*, there exists a good assignment* $\beta : V \to \{0, 1\}$ *with* $|C_I(\beta)\Delta\mathcal{P}| \leq k$ *and* $|\{v \in V \mid \alpha(v) \neq \beta(v)\}| \leq rk$*.*

Proof. Choose a good β with $|C_I(\beta)\Delta\mathcal{P}| \leq k$, and if there are multiple ones, choose the one with the smallest hamming distance to α.

For any v with $\alpha(v) \neq \beta(v)$, consider assignment β' that differs from β only on v. β' has smaller Hamming distance to α than β, so either β' is not good, or β' is good but does not satisfy the promise. Both cases require that some clause $C \in \mathcal{C}$ is satisfied in β but violated in β'. Since all clauses are AND clauses and $\beta'(v) = \alpha(v)$, such clause C is violated in α. Therefore, all variables contributing to the hamming distance are incident to some clauses in $C_I(\beta)\Delta\mathcal{P}$, and the statement follows from the fact that $|C_I(\beta)\Delta\mathcal{P}| \leq k$ and that the arity of all relations in $\cup_{r' \leq r} r'$AND is no more than r. ◀

Now we use the (derandomized) color coding technique along with the algorithm of Lemma 11 for Maximum Induced Subhypergraph with Vertex Weights to solve our problem ImproveMaxCSP($\cup_{r' \leq r} r'$AND).

▶ **Theorem 14.** *There is an algorithm solving* ImproveMaxCSP($\cup_{r' \leq r} r'$AND) *in* $2^{O(k \log k)} n^{O(1)}$ *time when the input instance satisfies an additional promise that* $\mathcal{P}$ *is satisfiable.*

Proof. First use Theorem 12 with $a = b = rk$ to obtain a family of at most $2^{O(k \log k)} \log n$ colorings with binary labels for variables. The algorithm then produces an assignment for each coloring and reports the best among them. In the following we fix a particular coloring.

Let L_1 be the set of variables with label 1. Construct a binary relation $\sim$ on L_1, where $u \sim v$ if $u = v$ or some clause $C \in \mathcal{P}$ is incident to both u and v. Since $\sim$ is reflexive and symmetric, its transitive closure is an equivalence relation. Denote $L_1/\sim$ as the quotient set of the equivalence relation which includes all equivalence classes of the transitive closure. The intuition is that, the algorithm will produce the answer from α by flipping several variables with label 1, and variables in one equivalence class are considered a group and would be flipped or kept at the same time.

We then construct a hypergraph $H(V_H, E_H)$ with weight function $w : V_H \to \mathbb{Z}$ as follows:

- For each equivalence class in $L_1/\sim$, introduce a vertex v in V_H, whose weight $w(v)$ equals to the number of clauses in $\mathcal{P}$ incident to any variable in the equivalence class. Intuitively, the weight measures that how many clauses we will lose if we flip this equivalence class.
- For each clause $C \in \mathcal{C}_I \backslash \mathcal{P}$, if C can be satisfied by flipping L_1 from α, introduce a hyperedge incident to all equivalence classes containing incident vertices of C. So the clause will be satisfied if we flip all equivalence classes incident to this hyperedge.

Finally, the algorithm uses the algorithm for Maximum Induced Subhypergraph with Vertex Weights proposed in Lemma 11 to find $V_0 \subseteq V_H$ that maximizes $|E(H(V_0))| - \sum_{v \in V_0} w(v)$, and returns the assignment produced by flipping in α all variables belonging to equivalence classes in V_0.

The algorithm runs in $2^{O(k \log k)} n^{O(1)}$ time since the whole process can be done in polynomial time except trying all $2^{O(k \log k)} n^{O(1)}$ colorings produced by Theorem 12. Now we show that the algorithm produces a good assignment for instances satisfying the promise

that some good assignment is close to the $\mathcal{P}$. Since we finally choose the best assignment among all colorings, it is sufficient to show that there exists some coloring that produces a good assignment.

Fix any good assignment β with $|C_I(\beta)\Delta\mathcal{P}| \leq k$ and $|\{v \in V \mid \alpha(v) \neq \beta(v)\}| \leq rk$, where α is an assighnment such that $\mathcal{P} = \mathcal{C}_I(\alpha_0)$. According to Lemma 13 such β exists. Define $N(C_I(\beta)\Delta\mathcal{P})$ to be all variables incident to any clause in $C_I(\beta)\Delta\mathcal{P}$. Define $V_0 = \{v \in N(C_I(\beta)\Delta\mathcal{P}) \mid \alpha(v) = \beta(v)\}$ and $V_1 = N(C_I(\beta)\Delta\mathcal{P})\backslash V_0$. Note that both $|V_0|, |V_1| \leq rk$.

According to Theorem 12, there exists a coloring assigning V_0 to 0 and V_1 to 1. We consider the behavior of the algorithm when trying this coloring.

We first claim that, there exists a set of equivalence classes in $L_1/\sim$ whose union is exactly V_1. This is equivalent to the statement that there is no $u \in V_1$ and $v \in L_1\backslash V_1$ such that $u \sim v$. Suppose $u \sim v$ and $u \in V_1$, then some $C \in \mathcal{P}$ is incident to both of them. Since C is incident to $u \in V_1$ which satisfies $\alpha(u) \neq \beta(u)$, C is violated by β, so $C \in \mathcal{C}_I(\beta)\Delta\mathcal{P}$. Then $v \in N(C_I(\beta)\Delta\mathcal{P})$, which finishes the proof since it implies $v \notin L_1\backslash V_1$.

The previous statement shows that, there exists some $V^\star \subseteq V_H$, such that by flipping from α all variables in equivalence classes corresponding to $V^\star$, one can produce β. We further show the following two claims:

- For any $V \subseteq V_H$, suppose α' is produced by flipping from α all variables in every equivalence class in V, then $cost_I(\alpha) - cost_I(\alpha') \leq |E(H(V))| - \sum_{v\in V} w(v)$. Notice that the latter is the objective of the Maximum Induced Subhypergraph with Vertex Weights problem. In other words, the cost of α' would not be underestimated. To see this, consider clauses in $C_I(\alpha)\Delta C_I(\alpha')$. Clauses satisfied in α but unsatisfied in α' are those belonging to $\mathcal{P}$ and incident to V, counted exactly by $\sum_{v\in V} w(v)$. Clauses satisfied in α' but unsatisfied in α are partially counted by $|E(H(V))|$, since it does not include satisfied clauses in α' violated by the assignment produced from flipping L_1 in α. However, all hyperedges in $E(H(V))$ correspond to clauses satisfied in α' but unsatisfied in α.
- $cost_I(\alpha) - cost_I(\beta) = |E(H(V^\star))| - \sum_{v\in V^\star} w(v)$, i.e. the cost of β is correctly estimated by the Maximum Induced Subhypergraph with Vertex Weights objective. We only need to show that $E(H(V^\star))$ contains all clauses satisfied in β but unsatisfied in α. Since the coloring colors V_0 with 0 and V_1 with 1, all clauses satisfied in β but unsatisfied in α have label-1 incident variables inside $V_1 \subseteq L_1$, so the clauses will introduce hyperedges in H whose scope is inside $V^\star$, so they are all counted by $|E(H(V^\star))|$.

The previous two claims imply that, $V^\star$ is one of the optimal solutions to the Maximum Induced Subhypergraph with Vertex Weights problem for the constructed hypergraph $5H$, and any optimal solution of the Maximum Induced Subhypergraph with Vertex Weights problem corresponds to a good assignment to the CSP instance, which finishes the proof. ◀

4 FPT Algorithms for 2AE

In this section we prove Theorem 7.

4.1 Preliminaries

We recall useful definitions about graphs. Let $G = (V, E)$ be a multigraph, where parallel edges and loops are allowed. For two disjoint vertex sets $A, B \subseteq V$, we denote by $E(A, B)$ the set of edges between A and B. When (A, B) is a partition, then $E(A, B)$ is the set

of edges in the corresponding cut. For a vertex subset $V' \subseteq V$, we denote by $G(V')$ the subgraph of G induced by V'. We write $E(G(V'))$ for the set of edges in $G(V')$. For a set $E' \subseteq E$, we denote by $G \setminus E'$ the multigraph $(V, E \setminus E')$. For $S \subseteq V$, we let $N(S) := \{v \in V \setminus S : \exists u \in S \text{ such that } (u, v) \in E\}$ be the open neighborhood of S. We say two cuts (A, B) and (X, Y) are k-close if $|E(A, B)\Delta E(X, Y)| \leq k$. When the graph is clear in the context, we define $n = \max(|V|, |E|)$ to be the size of the graph.

Notice that we can see clauses as edges connecting variables, assigning boolean values to variables as partitioning the variable set into two sets, and the equality and non-equality constraints correspond to uncut and cut constraints respectively. This motivates us to introduce the following graph problem. Given a graph $G = (V, E)$, an edge type function $t : E \to \{0, 1\}$ and a partition (A, B) of V, define $C(A, B) = (t^{-1}(1) \cap E(A, B)) \cup (t^{-1}(0) \cap (E \setminus E(A, B))$ to be the set of edges satisfied by (A, B), where edges of type 1 need to be separated, while edges of type 0 should be inside some vertex set.

Problem: ImproveMaxCut-Uncut
Input: A connected multigraph $G(V, E)$, an integer k, $\mathcal{P} \subseteq E$, and an edge type function $t : E \to \{0, 1\}$.
Parameter: k
Promise: There exists some (A, B) that maximizes $|C(A, B)|$ and satisfies $|C(A, B)\Delta\mathcal{P}| \leq k$.
Output: Any partition (A, B) of vertices that maximizes $|C(A, B)|$. (Notice that we do not require $|C(A, B)\Delta\mathcal{P}| \leq k$ for the output.)

Clearly, if the given graph for ImproveMaxCut-Uncut is not connected, we can solve each connected component separately, so connectivity of G does not lose any generality for the problem. The following claim is straightforward.

▷ Claim 15. ImproveMaxCSP(2AE) and ImproveMaxCut-Uncut are equivalent.

In the following, we present the algorithm for ImproveMaxCut-Uncut.

4.2 From Edge Solution to Vertex Solution

We first apply a pre-processing algorithm to the given ImproveMaxCut-Uncut instance, intending to make it in accordance with some partition of the vertex set.

▶ **Lemma 16.** *There exists an algorithm that given a ImproveMaxCut-Uncut instance, runs in $4^k n^{O(1)}$ time and finds a bipartition (A, B) of the vertex set V such that there exists a* good *partition $(A^\star, B^\star)$ which satisfies $|C(A^\star, B^\star)\Delta\mathcal{P}| \leq k$ and $|C(A^\star, B^\star)\Delta C(A, B)| \leq 3k$.*

Proof. Fix any $(A^\star, B^\star)$ satisfying $|C(A^\star, B^\star)\Delta\mathcal{P}| \leq k$. We first find the largest subset $\mathcal{P}_1 \subseteq \mathcal{P}$ that can be simutaneously satisfied by some bipartition using a $4^k n^{O(1)}$-time algorithm for MinCSP(2AE) [7]: Since $C(A^\star, B^\star) \cap \mathcal{P}$ is one possible solution, this means that $|\mathcal{P}_1| \geq |C(A^\star, B^\star) \cap \mathcal{P}| \geq \max(\mathcal{P}, |C(A^\star, B^\star)|) - k$. Let (A, B) be any bipartition satisfying $\mathcal{P}_1$. Consider $C(A, B)$: It contains all edges in $\mathcal{P}_1$ and perhaps more. However, since the size cannot be larger than $|C(A^\star, B^\star)|$, there are at most k edges in $C(A, B) \setminus \mathcal{P}_1$. Now we have

$$|C(A, B)\Delta C(A^\star, B^\star)| \leq |\mathcal{P}\Delta C(A^\star, B^\star)| + |\mathcal{P} \setminus \mathcal{P}_1| + |C(A, B) \setminus \mathcal{P}_1| \leq 3k.$$

This completes the proof. ◀

From Lemma 16, we can set $\mathcal{P}$ to be such $C(A, B)$ and triple k to change the original ImproveMaxCut-Uncut instance to an equivalent ImproveMaxCut-Uncut instance, whose $\mathcal{P}$ is given by some $C(A, B)$ for a vertex partition (A, B).

4.3 Algorithms for Instances with Vertex Solution

In the rest of this section, we prove the following theorem:

▶ **Theorem 17.** *There exists an algorithm that, given a ImproveMaxCut-Uncut instance $(G(V, E), k, \mathcal{P}, t)$ where $\mathcal{P} = C(A, B)$ for some partition (A, B) of V, finds a partition (X, Y) that maximizes $|C(X, Y)|$ in time $2^{O(k^2)}n^{O(1)}$.*

We first show how this theorem implies the main theorem.

Proof of Theorem 7. According to Claim 15, we only need to find such algorithm for ImproveMaxCut-Uncut. For a ImproveMaxCut-Uncut instance, we first apply the algorithm stated in Lemma 16 to produce an instance equivalent to the input whose $\mathcal{P}$ is given by some $C(A, B)$ for a vertex partition. Then we apply the algorithm stated in Theorem 17 to solve the new instance. The runtime is $4^k n^{O(1)} + 2^{O(k^2)}n^{O(1)} = 2^{O(k^2)}n^{O(1)}$. ◀

Recall that we are given a vertex partition (A, B) as the solution, while there exists a good partition (X, Y) with $|C(A, B)\Delta C(X, Y)| \leq k$. Let $AX := A \cap X$ and define AY, BX, BY similarly.

Our high-level approach follows the framework first introduced by Kawarabayashi and Thorup [10] for Minimum k-cut and its refinement by Chitnis et al. [4]. Let us define the following *terminal version* of the problem. We will fix k to be the parameter of the initial instance, which *remains invariant throughout the recursive calls to the terminal version.*

Problem: ImproveMaxCut-Uncut Terminal Version
Input: A connected multigraph $G(V, E)$, an integer $k' \leq k$, a $\mathcal{P} \subseteq E$, an edge type function $t : E \to \{0, 1\}$, a set of terminals $T \subseteq V$ with $|T| \leq 2k$ (Notice that it is k rather than k') and a *marked edge set* $M \subseteq E$.
Parameter: k'
Promise:
- $\mathcal{P} = C(A, B)$ for some partition (A, B);
- M is a matching allowing parallel edges: Any two edges in M are either completely disjoint or parallel copies of each other.
- There exists a good cut (X, Y) with $|C(A, B)\Delta C(X, Y)| \leq k'$ and $M \subseteq E(X, Y) \cap E(A, B)$.

Requirement: For *each* function $f : T \to \{X, Y\}$ and $0 \leq k'' \leq k$, output a cut $(X_{f,k''}, Y_{f,k''})$ satisfying all the following or output nothing:
1. consistent with f: $f(v) = X$ iff $v \in X_{f,k''}$ for all $v \in T$,
2. consistent with M: $M \subseteq E(A, B) \cap E(X_{f,k''}, Y_{f,k''})$,
3. k''-close to (A, B): $|C(A, B)\Delta C(X_{f,k''}, Y_{f,k''})| \leq k''$,

For any (X, Y) which is a good cut that satisfies $|C(A, B)\Delta C(X, Y)| \leq k''$ and $M \subseteq E(X, Y) \cap E(A, B)$ for some $k'' \leq k'$, the output $(X_{f^*,k''}, Y_{f^*,k''})$, where $f^\star$ is consistent to (X, Y), should additionally be a good cut (not necessarily equal to (X, Y)).

Clearly, ImproveMaxCut-Uncut is a special case with $M = T = \varnothing$ after doing the pre-processing in Lemma 16. In the following we introduce the algorithm solving ImproveMaxCut-Uncut Terminal Version.

Let $q := k^2 2^{2k+2}$. Before introducing the algorithm, we require one additional definition.

▶ **Definition 18** ((k,q)-cut)**.** Let $G = (V, E)$ be a multigraph and let $M \subseteq E$ be the set of marked edges which is a matching allowing parallel edges. A cut $L \subseteq V$ is called (k,q)-balanced (more simply a (k,q)-cut) if the following conditions are met.

- $|E(L, V \setminus L)| \leq k$.
- Both $G(L)$ and $G(V \setminus L)$ are connected (using both edges inside and outside M).
- Both $G(L)$ and $G(V \setminus L)$ contains at least q non-marked edges (edges outside M).

Our algorithm follows a simple win-win strategy. If there is no (k,q)-cut, a simple color-coding algorithm will solve the terminal problem. If there is a (k,q)-cut, by recursively solving a smaller subproblem, one can reduce the number of unmarked edges. Combining this with the algorithm to compute a (k,q)-cut, one can solve the terminal version in FPT time.

Now we present each of the three parts. We start by directly solving the case where there is no (k,q)-cut, using the (derandomized) coloring coding technique.

▷ Claim 19. One can solve ImproveMaxCut-Uncut Terminal Version in $2^{O(k \log q)} n^{O(1)}$ time when there is no (k,q)-cut in the instance.

Proof. Given an ImproveMaxCut-Uncut Terminal Version instance $I = (G, k', \mathcal{P} = C(A,B), t, T, M)$, the algorithm considers every $f : T \to \{X, Y\}$ and $0 \leq k'' \leq k'$ and does the following. Assume that f and k'' is consistent with some good cut (X, Y), because apart from running time, the requirement only concerns for the correct f and k''.

Note that the set of edges $S = C(A,B)\Delta C(X,Y) = E(A,B)\Delta E(X,Y) = E(AX, BX) \cup E(AY, BY) \cup E(AX, AY) \cup E(BX, BY)$ forms a cut between $L := AX \cup BY$ and $R := AY \cup BX$, no matter how t assigns type to edges. Since G is connected and $|S| \leq k''$, $G \setminus S$ has at most $k'' + 1$ connected components.

Assume towards a contradiction that there are two connected components L', R' of $G \setminus S$ such that $L' \subseteq L$ and $R' \subseteq R$ and both $G(L')$ and $G(R')$ have at least q non-marked edges. Then the minimum cut separating L' and R' in G will cut at most $k'' \leq k$ edges where the two resulting parts are connected and contain L' and R' respectively, which contradicts the fact that there is no (k,q)-cut. [3]

Therefore, at least one of L and R (say R) has the property that every connected component of $G \setminus S$ contained in R has at most q non-marked edges. Since marked edges form a matching, the number of vertices in such a component can be at most $2q + 2$, so we have $|R| = O(qk)$.

Let $\{R_i\}_{i=1}^{\ell}$ be the connected components of $G(R)$. Define $a_i^+ = |C(A\Delta R_i, B\Delta R_i) \setminus C(A,B)|$ and $a_i^- = |C(A,B) \setminus C(A\Delta R_i, B\Delta R_i)|$. Notice that $|C(X,Y)| = |C(A,B)| + \sum_{i=1}^{\ell}(a_i^+ - a_i^-)$ and $|C(X,Y)\Delta C(A,B)| = \sum_{i=1}^{\ell}(a_i^+) + (a_i^-) \leq k'$. Guessing the correct values of ℓ and a_i^+, a_i^- takes at most $k^{O(k)}$ time.

Let $N = N(R)$ be the open neighbor of R in G. Note that $|N| \leq |E(R, V \setminus R)| = |C(A,B)\Delta C(X,Y)| \leq k''$. Use Theorem 12 with $a = |R|, b = k$ to have a family of at most $2^{O(k \log q)} \log n$ colorings that color each vertex using $\{0, 1\}$. There exists a coloring χ that colors R with 1 and colors N with 0. For each $i \in [\ell]$, find all candidates $R_i' \subseteq V$ that is *identical* to R_i with respect to the guessed information about R_i. Formally, R_i' is a connected component of G after deleting all vertices of color 0, and in line with the guessing a_i^+ and a_i^-.

For the correct coloring χ, such R_i' exists since R_i satisfies the condition. Also, finding all such candidates R_i' is easy in polynomial time since one only needs to scan connected components after deleting color-0 vertices.

[3] We use the fact that any minimum st-cut in a connected graph have both sides connected.

For any $R'_1, \cdots, R'_\ell$ that are disjoint, since R'_i and $N(R'_j)$ are disjoint for any $i, j \in [\ell]$, the set of edges newly cut/uncut by updating $A \leftarrow A\Delta R'_i$ and $B \leftarrow B\Delta R'_i$ are disjoint for all i. Therefore, letting $R' := \cup_i R'_i$ and letting $A' \leftarrow A\Delta R'$ and $B' \leftarrow B\Delta R'$ the resulting partition has $|C(A', B')| = |C(A, B)| + \sum_i (a_i^+ - a_i^-) = |C(X, Y)|$ and $|C(A', B')\Delta C(A, B)| = \sum_i (a_i^+ + a_i^-) \leq k''$, so the solution is good and $k''-$close to (A, B).

We finally need the solution (A', B') to be consistent with both f and M. For any vertex $v \in T$, if v has color 0, then $v \in A$ if and only if $v \in X$ (recall $R = AY \cup BX$), otherwise by including v in R, one can change the belonging of v. So the consistency of f requires certain components to be included into or excluded from R'_i. For $(u, v) \in M$, if u and v has the same color then $M \in E(A, B) \cap E(X, Y)$ if and only if $M \in E(A, B)$ which is always satisfied, otherwise $M \in E(A, B) \cap E(X, Y)$ if and only if the label-1 vertex is not in R, so consistency of M requires certain components to be excluded from R'_i.

The consistency constraints are all in the form of "removing a component from the candidates" or "impose certain components to be selected by R'_i", which is easy to find out and handle in polynomial time.

We finally calculate the running time of the algorithm. The algorithm first tries all possible (f, k'') in $2^{O(k)}$ time, tries all colorings in $2^{O(k \log q)} n^{O(1)}$ time, and finally guesses the correct values of ℓ and a_i^+, a_i^- in $2^{O(k \log k)}$ time. All other processes are in polynomial time. So the total running time is $2^{O(k \log q)} n^{O(1)}$. ◁

Next we show how to compute a (k, q)-cut if it exists.

▷ Claim 20. There exists an algorithm that runs in time $2^{O(k \log q)} n^{O(1)}$ and returns a (k, q)-balanced cut for a connected graph if it exists.

Proof. Suppose that $L \subseteq V$ is a (k, q)-cut and $R = V \setminus L$. We first show that there exists $E_L \subseteq E(L)$ that induces a connected subgraph and has at most $O(q)$ edges but at least q non-marked edges. Consider a procedure where we start from $V_L = \{v\}$ for an arbitrary vertex $v \in L$, and iteratively add one arbitrary vertex $u \in L \cap N(V_L)$ to V_L until $E(G(V_L))$ contains at least q non-marked edges. Since $E(G(V_L))$ is connected and increased by at least one while marked edges form a matching, $|V_L| \leq 2q + 2$. Then we choose $E_L \subseteq E(G(V_L))$ that contains at least q non-marked edges while (V_L, E_L) is connected and $|E_L| \leq O(q)$. Define V_R and E_R similarly.

Use Theorem 12 with $a = O(q)$ and $b = k$ to try at most $2^{O(k \log q)} \log n$ colorings of edges with 2 colors. One coloring χ colors $E(L, R)$ with 0 and $E_L \cup E_R$ with 1. After deleting all edges of color 0, there are two connected components V'_L and V'_R such that $E_L \subseteq E(G(V'_L))$ and $E_R \subseteq E(G(V'_R))$. Note that $V'_L \neq V'_R$ because L and R are separated by removing edges of color 0.

Then, trying every pair (U', V') of connected components after deleting 0-colored edges and computing the minimum cut separating U' and V' in the original graph will yield a cut of size at most k where each side is connected has at least q non-marked edges. We use the fact that in a connected graph, any minimum s-t cut has both of its sides connected. ◁

Finally, we prove that the existence of a (k, q)-cut, combined with a recursion, allows us to make progress by reducing the number of unmarked edges. Let $T(m)$ be the running time of our algorithm to solve the terminal version with m unmarked edges.

▷ Claim 21. Suppose there exists a (k, q)-cut, in time $T(\ell) + n^{O(1)}$, one can reduce the current instance to another ImproveMaxCut-Uncut Terminal Version instance with at least $\ell - q/2$ fewer unmarked edges for some $\ell \in [q, m - q]$.

Proof. Let (L, R) be a (k, q)-cut. Without loss of generality, suppose that L has at most k terminals from T, and let $T_L \subseteq L$ be the set of vertices that have an edge to R. Letting $T' = (T \cap L) \cup T_L \subseteq L$ be the new set of terminals for L, we know that $|T'| \leq 2k$. Let ℓ be the number of unmarked edges in L and m be the total number of unmarked edges.

Recursively solve the terminal version with input $(G(L), (L \cap A, L \cap B), k', T', M \cap E(G(L)))$. For each $f' : T' \to \{X, Y\}$ and $0 \leq k'' \leq k'$, it returns a partition $(X_{f',k''}, Y_{f',k''})$ of L that is (1) consistent with f', (2) consistent with $M \cap E(G(L))$, and (3) k''-close to $(L \cap A, L \cap B)$ (or nothing).

Let $f^* : T' \to \{X, Y\}$ be the correct guessing with respect to the good partition (X, Y) and k^* be the correct closeness parameter $|C(A \cap L, B \cap L)\Delta C(X \cap L, Y \cap L)|$ in L. We update (X, Y) with $X \leftarrow (X \cap R) \cup X_{f^*,k^*}$ and $Y \leftarrow (Y \cap R) \cup Y_{f^*,k^*}$. Note that the new (X, Y) is still a good cut since $(X_{f^*,k^*}, Y_{f^*,k^*})$ is good within L and the interaction between L and R only depends on the terminals T', where the previous and new solutions agree. By the guarantee (2) and (3) in the above paragraph, the new (X, Y) is still k'-close to (A, B) and consistent with M.

Since L has at least $q = k^2 2^{2k+2}$ non-marked edges and there are at most $2^{2k}(k+1)$ possible values for f' and k'', we have $|E(L) \setminus (\cup_{f',k''}(C(A \cap L, B \cap L)\Delta C(X_{f',k''}, Y_{f',k''}))| \geq \ell - q/2$. These edges do not belong to $C(A, B)\Delta C(X, Y) = E(A, B)\Delta E(X, Y)$ for sure. For each such edge e, perform the following operation.

- If $e \notin E(A, B)$, which means that e is not cut in the current solution, this means that e is not cut in the good solution too. Contract it.
- If $e \in E(A, B)$, mark it. To ensure that the set of marked edges M is a matching, if $e = (u, v), f = (v, w)$ are both in M with $u \neq w$, then we know that u, w will be in the same side in the good solution, so we can contract them.

By doing this, the number of unmarked edges is decreased by at least $\ell - q/2$. ◁

With the three main claims, we finish the proof of Theorem 17.

Proof of Theorem 17. From the definition of the terminal version, given graph G and the current solution (A, B), running the terminal version with $T = M = \varnothing$ and $k' = k$ we can find a good cut given the promise $|C(A, B)\Delta C(X, Y)| \leq k$ for some (possibly different) good cut (X, Y).

Let us analyze the running time. As defined previously, let $T(m)$ be the running time of our algorithm with m unmarked edges. Since the graph is connected and marked edges form a matching, we have $m \geq \frac{n}{2}$. Our algorithm first uses Claim 20 to find a (k, q)-cut in time $2^{O(k \log q)} n^{O(1)}$. If there is none, it uses Claim 19 to solve the terminal version in time $2^{O(k \log q)} n^{O(1)}$. Otherwise, it uses Claim 21 to reduce the number of unmarked edges by $\ell - q/2$ in time $T(\ell) + n^{O(1)}$, for some $\ell \in [q, m - q]$. Therefore, we have the following recurrence relation:

$$T(m) \leq 2^{O(k \log q)} n^{O(1)} + \max_{\ell \in [q, m-q]} \left(T(\ell) + T(m - (\ell - q/2)) + n^{O(1)} \right).$$

Using $m \in \left[\frac{n}{2}, n^2\right]$, $T(m) \leq 2^{O(k \log q)} n^{O(1)}$ satisfies the above. ◀

References

1 Aditya Anand, Euiwoong Lee, and Amatya Sharma. Min-CSPs on complete instances. In *Proceedings of the 2025 ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 2025.

2 Edouard Bonnet, László Egri, and Dániel Marx. Fixed-parameter approximability of boolean mincsps. In *24th European Symposium on Algorithms (ESA 2016)*, page 246, 2016.

3 Nicolas Bousquet, Jean Daligault, and Stéphan Thomassé. Multicut is fpt. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 459–468, 2011. doi:10.1145/1993636.1993698.

4 Rajesh Chitnis, Marek Cygan, MohammadTaghi Hajiaghayi, Marcin Pilipczuk, and Michał Pilipczuk. Designing fpt algorithms for cut problems using randomized contractions. *SIAM Journal on Computing*, 45(4):1171–1229, 2016. doi:10.1137/15M1032077.

5 Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. Minimum bisection is fixed parameter tractable. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 323–332, 2014. doi:10.1145/2591796.2591852.

6 Marek Cygan, Marcin Pilipczuk, Michał Pilipczuk, and Jakub Onufry Wojtaszczyk. On multiway cut parameterized above lower bounds. *ACM Transactions on Computation Theory (TOCT)*, 5(1):1–11, 2013. doi:10.1145/2462896.2462899.

7 Konrad K Dabrowski, Peter Jonsson, Sebastian Ordyniak, George Osipov, and Magnus Wahlström. Almost consistent systems of linear equations. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3179–3217. SIAM, 2023. doi:10.1137/1.9781611977554.CH121.

8 Michael R. Fellows, Fedor V. Fomin, Daniel Lokshtanov, Frances Rosamond, Saket Saurabh, and Yngve Villanger. Local search: Is brute-force avoidable? *Journal of Computer and System Sciences*, 78(3):707–719, 2012. In Commemoration of Amir Pnueli. doi:10.1016/j.jcss.2011.10.003.

9 Serge Gaspers, Eun Jung Kim, Sebastian Ordyniak, Saket Saurabh, and Stefan Szeider. Don't be strict in local search! In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 26, pages 486–492, 2012. doi:10.1609/AAAI.V26I1.8128.

10 Ken-ichi Kawarabayashi and Mikkel Thorup. The minimum k-way cut of bounded size is fixed-parameter tractable. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 160–169. IEEE, 2011. doi:10.1109/FOCS.2011.53.

11 Eun Jung Kim, Stefan Kratsch, Marcin Pilipczuk, and Magnus Wahlström. Solving hard cut problems via flow-augmentation. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 149–168. SIAM, 2021. doi:10.1137/1.9781611976465.11.

12 Eun Jung Kim, Stefan Kratsch, Marcin Pilipczuk, and Magnus Wahlström. Directed flow-augmentation. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 938–947, 2022. doi:10.1145/3519935.3520018.

13 Eun Jung Kim, Stefan Kratsch, Marcin Pilipczuk, and Magnus Wahlström. Flow-augmentation iii: Complexity dichotomy for boolean csps parameterized by the number of unsatisfied constraints. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3218–3228. SIAM, 2023. doi:10.1137/1.9781611977554.CH122.

14 Andrei Krokhin and Dániel Marx. On the hardness of losing weight. *ACM Transactions on Algorithms (TALG)*, 8(2):1–18, 2012. doi:10.1145/2151171.2151182.

15 Tommaso Lanciano, Atsushi Miyauchi, Adriano Fazzone, and Francesco Bonchi. A survey on the densest subgraph problem and its variants. *ACM Computing Surveys*, 56(8):1–40, 2024. doi:10.1145/3653298.

16 Daniel Lokshtanov, Saket Saurabh, and Vaishali Surianarayanan. A parameterized approximation scheme for min k-cut. *SIAM Journal on Computing*, (0):FOCS20–205, 2022.

17 Dániel Marx. Parameterized complexity of constraint satisfaction problems. *computational complexity*, 14:153–183, 2005. doi:10.1007/S00037-005-0195-9.

18 Dániel Marx and Igor Razgon. Fixed-parameter tractability of multicut parameterized by the size of the cutset. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 469–478, 2011. doi:10.1145/1993636.1993699.

19 Dániel Marx and Ildikó Schlotter. Stable assignment with couples: Parameterized complexity and local search. *Discrete Optimization*, 8(1):25–40, 2011. Parameterized Complexity of Discrete Optimization. doi:10.1016/j.disopt.2010.07.004.

20 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.

21 NS Narayanaswamy, Venkatesh Raman, MS Ramanujan, and Saket Saurabh. Lp can be a cure for parameterized problems. In *29th International Symposium on Theoretical Aspects of Computer Science (STACS 2012)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2012. doi:10.4230/LIPIcs.STACS.2012.338.

22 Venkatesh Raman, MS Ramanujan, and Saket Saurabh. Paths, flowers and vertex cover. In *Algorithms–ESA 2011: 19th Annual European Symposium, Saarbrücken, Germany, September 5-9, 2011. Proceedings 19*, pages 382–393. Springer, 2011.

23 Igor Razgon and Barry O'Sullivan. Almost 2-sat is fixed-parameter tractable. In *Automata, Languages and Programming: 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part I 35*, pages 551–562. Springer, 2008.

24 Neil Robertson and Paul D Seymour. Graph minors. xiii. the disjoint paths problem. *Journal of combinatorial theory, Series B*, 63(1):65–110, 1995. doi:10.1006/JCTB.1995.1006.

25 Stefan Szeider. The parameterized complexity of k-flip local search for sat and max sat. *Discrete Optimization*, 8(1):139–145, 2011. doi:10.1016/J.DISOPT.2010.07.003.

Uniformity Within Parameterized Circuit Classes

Steef Hegeman
Leiden University, The Netherlands

Jan Martens
Leiden University, The Netherlands

Alfons Laarman
Leiden University, The Netherlands

Abstract

We study uniformity conditions for parameterized Boolean circuit families. Uniformity conditions require that the infinitely many circuits in a circuit family are in some sense easy to construct from one shared description. For shallow circuit families, logtime-uniformity is often desired but quite technical to prove. Despite that, proving it is often left as an exercise for the reader – even for recently introduced classes in parameterized circuit complexity, where uniformity conditions have not yet been explicitly studied. We formally define parameterized versions of linear-uniformity, logtime-uniformity, and FO-uniformity, and prove that these result in equivalent complexity classes when imposed on para-AC^0 and para-$\mathsf{AC}^{0\uparrow}$. Overall, we provide a convenient way to verify uniformity for shallow parameterized circuit classes, and thereby substantiate claims of uniformity in the literature.

2012 ACM Subject Classification Theory of computation → Complexity classes; Theory of computation → Circuit complexity; Theory of computation → Fixed parameter tractability

Keywords and phrases Parameterized complexity, circuit complexity, uniformity, descriptive complexity

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.27

Related Version *Full Version including Appendices*: `https://arxiv.org/abs/2509.09657`

1 Introduction

Circuit complexity theory analyzes problems through the (smallest) size and depth of Boolean circuits that solve them, linking logic, space complexity, and parallel complexity theory [16]. Because a Boolean circuit has a fixed number of inputs, the actual object of study is a circuit family, which contains a circuit for every input length n.

Without restrictions, circuit families are too expressive to be used as a model of computation, as non-uniform circuit families can even express non-computable functions. Hence, uniformity conditions are often imposed to ensure that the circuits in the family are in some sense easy to construct. With uniformity conditions, circuit families are a useful model in the study of, amongst others, parallel and descriptive complexity theory [16].

The main idea behind uniformity conditions is that, for a class of circuit families $\mathcal{C}$, the structure of a family's circuits themselves should be less complex than the problems that families in $\mathcal{C}$ can solve. For the class of constant-depth circuit families, this line of thinking has led to the very restrictive logtime-uniformity condition, requiring that the edge relation of the circuits in a family must be (simultaneously) decidable in logarithmic time.

Logtime computations are difficult to analyze or work with [11, 16]. Therefore, logtime-uniformity is rarely proved directly, but instead shown using results such as those of Barrington, Immerman, and Straubing [11]. They show that if a constant-depth circuit family is describable in terms of first-order logic, there is another, logtime-uniform constant-depth

© Steef Hegeman, Jan Martens, and Alfons Laarman;
licensed under Creative Commons License CC-BY 4.0

circuit family equivalent with it. Using this correspondence, one can show the existence of a logtime-uniform circuit family without directly dealing with the technical intricacies of logtime computations.

Recently, the interest in circuit complexity was renewed in the setting of parameterized complexity [1, 2, 4, 12, 3]. In parameterized complexity theory, problems are classified on a finer scale with respect to an additional parameter instead of just input size. In the context of circuit complexity this spawns new circuit classes, that identify problems which are in some sense parallel constant time decidable up-to some parameter [2].

Parameterized circuit classes require new parameterized uniformity conditions. Recent works [1, 2, 4, 12, 3] have adapted the logtime-uniformity of Barrington et al., though they are not very specific in their exact definitions. They generally refer to the complexity of computing single bits in the binary encoding of circuits, but it is not specified what the binary encoding is. Additionally, though the literature cites [11], it is not clear if the equivalence results of Barrington et al. carry over to the parameterized circuit classes.

In general, the literature often leaves uniformity claims as an exercise for the reader. To the best of our knowledge, only [1, Theorem 3.2, Appendix A] sets out to prove one of its uniformity claims. However, the proof appears to erroneously assume that logtime-uniformity gives a random-access Turing machine polylogarithmic time to decide the graph structure of the circuits, while it allows only logarithmic time.

Table 1 Informal overview of the parameterized uniformity notions used in the paper.

Condition	Question (for circuit $C_{n,k}$)	Decision complexity
linear-BD-uniform	$\langle$local question, $n, k\rangle$	$\mathsf{DTIME}(\log n + f(k))$
logtime-D-uniform	$\langle$local question, $\{0,1\}^n, \{0,1\}^k\rangle$	$\mathsf{DTIME}^{\mathsf{R}}(\log n + f(k))$
logtime-E-uniform	$\langle$path question, $\{0,1\}^n, \{0,1\}^k\rangle$	$\mathsf{DTIME}^{\mathsf{R}}(\log n + f(k))$
FO-D-uniform	$\langle$local question, $\{0,1\}^n, \{0,1\}^k\rangle$	para-FO in parameter k
FO-E-uniform	$\langle$path question, $\{0,1\}^n, \{0,1\}^k\rangle$	para-FO in parameter k

Contributions. We introduce and analyze uniformity conditions for parameterized circuit complexity. Specifically, we define a suitable notion of the logtime-uniformity used in the literature, and show it to be equivalent to more natural notions when imposed on the shallow parameterized circuit classes para-AC^0 and para-$\mathsf{AC}^{0\uparrow}$.

An overview of the discussed uniformity notions, formally defined in Sections 3 and 5, can be found in Table 1. We consider direct uniformity notions, which require the local structure of circuits to be easily decidable, and extended uniformity notions that also consider paths in the circuit. The direct logtime-D-uniformity is based on logtime-DCL-uniformity introduced in [11] and used in the reference work [16]. We believe our definition captures the more informal notions of parameterized logtime-uniformity found in [1, 2, 4, 12, 3]. For the classes para-AC^0 and para-$\mathsf{AC}^{0\uparrow}$, we show that these notions are equivalent to linear-BD-uniformity, a parameterized variant of U_D-uniformity from [14]. We introduce a parameterized variant of FO-uniformity from [11], and show that this too is equivalent to our earlier notions when applied to para-AC^0 and para-$\mathsf{AC}^{0\uparrow}$. Finally, we also show this for strict adaptations of extended logtime and first-order uniformity.

Our results allow for an easier analysis of uniformity for shallow parameterized circuit classes, and address uniformity claims in the literature.

Outline. Section 2 contains preliminaries on circuit complexity and known uniformity results. In Section 3, we formally define the direct uniformity conditions roughly described in Table 1. In Section 4, we show that imposing these different conditions leads to the same complexity class, both for para-AC^0 and para-$\mathsf{AC}^{0\uparrow}$. In Section 5, we show that this also holds for strict adaptations of extended uniformity conditions. In Section 6, we show how our work can be related to the descriptive complexity-based work on uniformity for $\mathsf{AC}[t(n)]$.

2 Preliminaries

In this section, unless otherwise noted, standard definitions in circuit complexity and parameterized complexity are based on the reference works [16] and [7] respectively.

2.1 Circuit families

A **circuit family** is a set of (Boolean) circuits $\{C_n : n \in \mathbb{N}\}$ such that C_n has n input gates (fan-in 0) and at least one output gate (fan-in 1, fan-out 0). The remaining (**internal**) gates are AND, OR, or NOT gates. A circuit's **size** is the number of gates in it, and its **depth** the length of the longest (input to output) path it contains. The size and depth of a circuit family C are functions s, d, where $s(n), d(n)$ are equal to the size and depth of C_n respectively.

We generally care about circuit families in which each circuit has one output gate. We then define the language recognized by the circuit family C as $\{x \in 2^* : C_{|x|}(x) = 1\}$. That is, given some binary string x, we take the circuit $C_{|x|}$ with $|x|$ input gates, set the input gates to the bits of x, and evaluate it. The word is accepted if and only if the output gate evaluates to 1. More generally, a circuit with n inputs and m outputs induces a binary function $2^n \to 2^m$. Circuit families induce a function $2^* \to 2^*$.

The **circuit class** AC^0 consists of all languages that are recognized by polynomially sized circuit families of constant depth. That is, Q is in AC^0 if there is a circuit family C with size $s(n) = O(n^c)$ and depth $d(n) = O(1)$, such that $Q = \{x : C(x) = 1\}$. It is important to note that all gates have unbounded fan-out, and the AND and OR gates unbounded fan-in.

2.2 Parameterized complexity

Parameterized complexity theory studies computational problems modulo (pre)computations on some input-based parameter. The parameter function is considered part of the problem.

▶ **Definition 1.** *A* **parameterized problem** (Q, κ) *consists of a language* $Q \subseteq 2^*$ *and a polynomial-time computable* **parameter function** $\kappa : 2^* \to \mathbb{N}$.

We follow the literature on shallow parameterized circuit complexity [5, 1] in additionally requiring κ to be FO-computable [9, Definition 2.5], equivalently that κ can be computed by a logtime-D-uniform circuit family of polynomial size and constant depth (cf. Definition 8). This requirement can be removed by modifying the definition of para-FO to being given $f(\kappa(x))$ as a precomputed value, as discussed in the full version (Appendix A).

Parameterized classes are often defined in terms of non-parameterized classes.

▶ **Definition 2** ([6]). *Let* $\mathcal{X}$ *be a complexity class. A parameterized problem* (Q, κ) *is in* $\mathrm{Para}(\mathcal{X})$ *if and only if there is a computable function* f *with* $\{\langle x, f(\kappa(x))\rangle : x \in Q\}$ *in* $\mathcal{X}$.

Classes such as parameterized linear time can also be given more explicitly.

▶ **Definition 3.** *We write* $\mathsf{DTIME}(t(n) + f(k))$ *for the class of parameterized problems* (Q, κ) *for which there is a multitape Turing machine that solves* $x \in Q$ *in* $O(t(|x|) + f(\kappa(x)))$ *time.*

The difference in style between these definitions lies in whether the precomputation is part of the input or given implicitly by means of extra time. In [6], it is shown that (Q, κ) is in $\mathsf{DTIME}(t(n) + f(k))$ for some computable f if and only if (Q, κ) is in para-$\mathsf{DTIME}(t(n))$, but there are no such results for the sublinear random-access Turing machines we use later.

2.3 Parameterized circuit complexity

Parameterized circuit complexity studies classes $\mathrm{Para}(\mathcal{X})$ for ordinary circuit complexity classes $\mathcal{X}$, as well as other classes defined in terms of parameterized circuit families.

▶ **Definition 4** ([1]). *A* **parameterized circuit family** C *is a set* $\{C_{n,k} : n, k \in \mathbb{N}\}$ *of circuits, such that each* $C_{n,k}$ *has* n *input gates. A parameterized problem* (Q, κ) *is decided by* C *if* $x \in Q \iff C_{|x|,\kappa(x)}(x) = 1$ *holds. Size and depth are now functions* $s(n, k)$ *and* $d(n, k)$*.*

The parameterized circuit class para-AC^0 is defined as the parameterized problems solved by constant-depth parameterized circuit families of size $O(f(k)n^c)$, where f is again some computable function that varies per circuit family. The class para-$\mathsf{AC}^{0\uparrow}$ is defined in terms of $O(f(k)n^c)$-sized circuits with depth $O(f(k))$ [1]. $\mathrm{Para}(\mathsf{AC}^0)$ is equal to para-AC^0 [5].

2.4 Uniformity

Uniformity conditions for shallow circuit classes are defined in terms of connection languages. These languages express the connections between the gates in the circuit family. To achieve this goal, the gates in each circuit are first identified by a numbering.

▶ **Definition 5.** *An* **admissible numbering** *of a circuit family* $C = \{C_n : n \in \mathbb{N}\}$ *of size* $s(n)$ *assigns a number to every gate in* C *so that:*

1. *within each* C_n*, no two gates have the same number,*
2. *the* n *input gates of* C_n *are numbered from 0 to* $n - 1$*,*
3. *the* m *output gates of* C_n *are numbered from* n *to* $n + m - 1$*, and,*
4. *gates in* C_n *are numbered polynomially in* $s(n)$*, that is, with numbers of* $O(\log s(n))$ *bits.*

To improve readability, we will assume that circuit families come with fixed admissible numberings (which we may choose freely), and often identify gates with their gate number.

▶ **Definition 6** ([11, 14]). *Let* C *be a circuit family. The* **direct connection language** $L_D(C)$ *consists of all strings of the form* $\langle G, a, p, z\rangle$ *where, writing* n *for* $|z|$*:*

1. G *is a gate in* C_n*;*
2. *if* $p = \epsilon$ *then* $a < 3$ *indicates the type of* G *(one of* AND*,* OR*,* NOT*);*
3. *otherwise, the* p*th gate feeding into* G *is* a*.*

The **binary direct connection language** $L_{BD}(C)$ *consists of all strings of the form* $\langle G, a, p, n\rangle$ *satisfying the same conditions, now with* n *written in binary. That is:*

$$L_{BD}(C) = \{\langle G, a, p, |z|\rangle : \langle G, a, p, z\rangle \in L_D(C)\}.$$

There is also an extended connection language $L_E(C)$, wherein p can also be a path. We will come back to this in Section 5. The encoding of the lists $\langle \ldots, \ldots \rangle$ in Definition 6 is addressed in Section 2.6. First, we introduce three direct uniformity notions, defined in terms of the decision complexity of direct connection languages.

Linear-time. In [14], Ruzzo introduced uniformity conditions for shallow circuit classes. The following definition is based on Ruzzo's U_D-uniformity.[1]

▶ **Definition 7.** *A circuit family C is* **linear-BD-uniform** *if $L_{BD}(C)$ is in* $\mathsf{DTIME}(n)$.

That is, a deterministic multitape Turing machine can decide $w \in L_{BD}(C)$ in time $O(|w|)$.

Logtime. Later uniformity notions were based on random-access Turing machines. A **random-access Turing machine** [16, Appendix A4] is a multitape Turing machine equipped with a special query tape and query state. Whenever the machine enters the query state, it obtains the value of the ath bit on the input tape, where a is the binary address written on the query tape, or an out-of-bounds response if a is beyond the length of the input (or not a number). Importantly, the query tape is not erased on query, and the input tape is read-only.

We write $\mathsf{DTIME}^{\mathsf{R}}(t(n))$ for the class of problems that can be solved in $O(t(n))$ time by a random-access Turing machine, and define $\mathsf{DTIME}^{\mathsf{R}}(t(n) + f(k))$ analogously to Definition 3.

▶ **Definition 8** ([11]). *Circuit family C is* **logtime-D-uniform** *if $L_D(C)$ is in* $\mathsf{DTIME}^{\mathsf{R}}(\log n)$.

Logtime-D-uniformity is arguably the most well-known uniformity condition for AC^0. Yet not much is known about $\mathsf{DTIME}^{\mathsf{R}}(\log n)$. Clearly $\mathsf{DTIME}^{\mathsf{R}}(\log n) \subseteq \mathsf{DTIME}(n \log n)$ [13], but it is to the best of our knowledge unknown whether this is strict. As far as we know, it is for example also open whether a random-access Turing machine can multiply two m-bit numbers in $O(m)$ time. What is known is that they can add m-bit numbers in $O(m)$ time, and determine the length of their input in $O(\log n)$ time [11, Lemma 6.1].

First-order definable. The class FO consists of the languages that are first-order definable in the predicates $\leq$ and Bit. That is, they are defined by a first-order sentence using symbols from $\{\forall, \wedge, \neg, \leq, \mathrm{Bit}, X\}$ as follows [11, 9]. A word w induces a first-order structure S_w whose domain consists of the numbers 0 to $|w|$, where: $\leq$ is interpreted as the standard order on natural numbers, $\mathrm{Bit}(i, j)$ is true if and only if the ith bit of j is 1, and $X(i)$ is true if and only if the ith bit of w is 1. A sentence ϕ then defines the language $\{w : S_w \models \phi\}$.

We may assume that the language also contains ternary predicates for addition and multiplication, as they are first-order definable in $\leq$ and Bit [11, 9]. We may also assume that it contains a constant symbol for the length of the input, and that we may quantify over (the binary representations of) numbers below $O(|w|^c)$, by using $O(c)$ variables [11, p. 293].

▶ **Definition 9** ([11]). *A circuit family C is* FO**-D-uniform** *if $L_D(C)$ is in* FO.

For polynomially-sized circuit families, it can be shown that linear-BD-uniformity implies logtime-D-uniformity, which in turn implies FO-D-uniformity. FO-D-uniformity is easier to work with than logtime-D-uniformity. In fact, there are many examples of languages for which the natural circuit families deciding them are quickly seen to be FO-D-uniform, but for which it is unknown whether they are logtime-D-uniform or linear-BD-uniform.

▶ **Example 10.** Consider problem $X = \{x : \text{the } \lfloor\sqrt{|x|}\rfloor\text{th bit of } x \text{ is } 1\}$. It is easy to see that X is in AC^0: there is a circuit family C where each C_n consists of essentially one wire from the $\lfloor\sqrt{n}\rfloor$th input gate to the output gate. It has only one admissible numbering, and its direct connection language is $\{\langle n, \lfloor\sqrt{n}\rfloor, 0, z\rangle : |z| = n\}$.

[1] Ruzzo requires $\mathsf{DTIME}(\log s(n))$ on a slightly different language, but this is equivalent for our cases.

The most difficult part in showing that this family is uniform is to show that we can somehow identify $\lfloor\sqrt{n}\rfloor$. For FO-D-uniformity, this is easy: $r = \lfloor\sqrt{n}\rfloor$ is defined by the formula $r \cdot r \leq n \wedge (\forall s > r)\ s \cdot s > n$. Showing that C is logtime-D-uniform appears to be more difficult, and we do not know whether this is the case. It is likely not linear-BD-uniform, as multiplication is conjectured to not be linear-time computable [8]. ⌟

2.5 Uniform AC^0

Rather than the uniformity of specific circuit families, we are usually more interested in whether an equivalent uniform family exists for the same complexity class.

▶ **Convention.** *Let $\mathcal{U}$ be a uniformity condition, and $\mathcal{C}$ a class of (parameterized) circuit families. If $\mathcal{X}$ is defined as the class of problems solved by families in $\mathcal{C}$, then* **$\mathcal{U}$-uniform $\mathcal{X}$** *is the subclass of problems solved by $\mathcal{U}$-uniform families in $\mathcal{C}$.*

From this perspective, uniformity conditions can be shown to be equivalent for AC^0.

▶ **Theorem 11** (Barrington, Immerman, Straubing [11])**.** *The following classes are equal:*
1. *logtime-D-uniform* AC^0
2. FO*-D-uniform* AC^0
3. FO

On the way to our main results we prove the following folklore extension of Theorem 11.

▶ **Theorem 12.** *Linear-BD-uniform* AC^0 *equals logtime-D-uniform* AC^0*.*

A stronger version, that logtime-D-uniformity is even equivalent with linear-BD-uniformity, is stated in [13, Section 4] and appears to be used implicitly in other work [15, Section 1.1]. However, this is not something we can prove. The argument in [13] seems to silently assume that $\mathsf{DTIME}^{\mathsf{R}}(n) = \mathsf{DTIME}(n)$ which we believe is an open problem.

We stress that the results above do not show that the uniformity conditions are equivalent. In particular, $\mathsf{DTIME}^{\mathsf{R}}(\log n)$ is strictly weaker than FO. It merely happens to be the case that the extra power of FO for constant-depth circuit family descriptions does not lead to circuit families with more computational power than any logtime-uniform circuit family.

2.6 Encoding lists

We define an encoding for the lists in Definition 6 as a generalization of the 2-tuple encoding used by Barrington et al. [11]. Each element $x \in 2^*$ is coded as $\delta(|x|)01x$, where $\delta(|x|)$ encodes the length of x in base $\{00, 11\}$ and 01 acts as a separator.

▶ **Definition 13.** *We encode $\langle x_0, \ldots, x_m\rangle$ as*

$$\delta(|x_0|)01x_0\,\delta(|x_1|)01x_1 \ldots \delta(|x_m|)01x_m$$

and define projection functions $\pi_i : 2^* \to 2^*$ by

$$\pi_i(x) = \begin{cases} x_i & \text{if } x \text{ is of the form } \langle x_0, \ldots, x_m\rangle \text{ with } i \leq m, \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

For example, $\langle 0, 00, 000\rangle$ is encoded as

$$\underbrace{11}_{\delta(|x_0|)}\ 01\ \underbrace{0}_{x_0}\ \underbrace{1100}_{\delta(|x_1|)}\ 01\ \underbrace{00}_{x_1}\ \underbrace{1111}_{\delta(|x_2|)}\ 01\ \underbrace{000}_{x_2}.$$

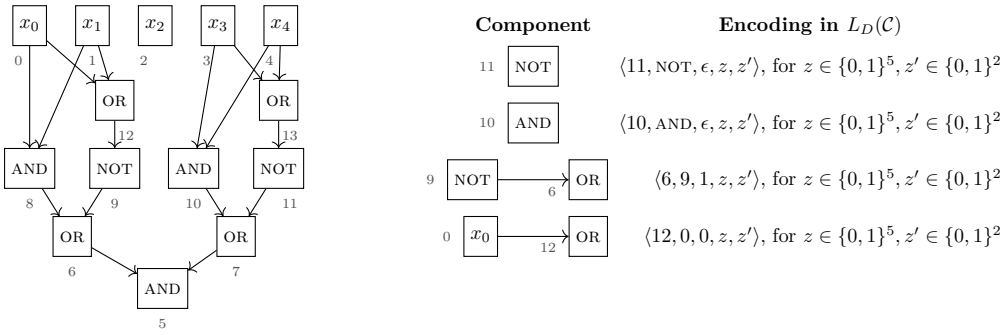


Figure 1 The circuit $C_{5,2}$ from the family C, and associated words $w \in L_D(C)$.

The encoding can be worked with in linear time, in random-access logarithmic time, and in first order logic with ordering and Bit. In particular, it can be checked whether a string encodes a list, the lengths of list items can be extracted, and the projection functions can be computed, provided that the relevant item is not too long (a logtime-machine cannot retrieve items of superlogarithmic length). Details can be found in the full version (Appendix B).

2.7 Parameterized uniformity

In [3, Proposition 6], Chen and Flum prove a uniform version of para-$\mathsf{AC}^0 = \text{Para}(\mathsf{AC}^0)$. Their proof carries over to our setting of parameterized linear-BD-uniformity we define in Section 3 as follows. In our reformulation we use that parameter functions κ are FO-computable.

▶ **Theorem 14** ([3]). *Linear-BD-uniform* para-AC^0 *is equal to* Para(*linear-BD-uniform* AC^0).

3 Direct parameterized uniformity

We first define uniformity notions for parameterized circuit families. Admissible numberings (Definition 5) carry over to parameterized circuit families in the straightforward way, and we again assume each parameterized family to come with such a numbering.

▶ **Definition 15.** *Let C be a parameterized circuit family. The* **direct connection language** *$L_D(C)$ consists of all strings of the form $\langle G, a, p, z, z' \rangle$ where, writing n for $|z|$ and k for $|z'|$:*

1. *G is a gate in $C_{n,k}$;*
2. *if $p = \epsilon$ then $a < 3$ and indicates the type of G (one of* AND, OR, NOT*);*
3. *otherwise, the pth gate feeding into G is the gate numbered a.*

The **binary direct connection language** *$L_{BD}(C)$ consists of all strings of the form $\langle G, a, p, n, k \rangle$ satisfying the same conditions, now with n and k written in binary. That is:*

$$L_{BD}(C) = \{\langle G, a, p, |z|, |z'| \rangle : \langle G, a, p, z, z' \rangle \in L_D(C)\}.$$

▶ **Example 16.** Consider the language

$$L = \{x_0 \dots x_{n-1} \mid \forall i \in [1, \lfloor \sqrt{n} \rfloor]. x_0 = x_{i-1} \text{ and } x_{n-i} = x_{n-1}\}.$$

Figure 1 shows the circuit $C_{5,2}$ from the circuit family C that computes the parameterized problem (L, κ), where the parameter function is defined by $\kappa(x) = \lfloor \sqrt{|x|} \rfloor$, together with some words from the direct connection language $L_D(C)$. ⌟

We now define uniformity conditions for parameterized circuit families. These are based on non-parameterized uniformity conditions. The difference is that parameterized uniformity conditions allow a precomputation on the same parameter function used for the circuit family. That is, where before we had $t(n)$ time to decide connections in circuits C_n, we now have $t(n) + f(k)$ time to decide connections in circuits $C_{n,k}$, for a computable f of choice.

We also adapt FO-D-uniformity. Write para-FO for Para(FO). The motivation for FO-D-uniformity for parameterized circuit families is that words $\langle G, a, p, z, z' \rangle$ relating to $C_{|z|,|z'|}$ should now be first-order expressible in terms of $|z|$ and $f(|z'|)$, where f is a computable function of choice. Again writing π_4 for the fifth projection function (so that $\pi_4(\langle G, a, p, z, z' \rangle) = z'$), this can equivalently be expressed as $(L_D(C), \pi_4)$ being in para-FO.

▶ **Definition 17.** *Let C be a parameterized circuit family. We call it*

linear-BD-uniform *if $(L_{BD}(C), \pi_4)$ is in $\mathsf{DTIME}(n + f(k))$ for some computable f, that is, on inputs w of the form $\langle G, a, p, n, k \rangle$ it takes time $O(|w| + f(k))$;*

logtime-D-uniform *if $(L_D(C), \pi_4)$ is in $\mathsf{DTIME}^{\mathsf{R}}(\log n + f(k))$ for some computable f, that is, on inputs w of the form $\langle G, a, p, z, z' \rangle$ it takes time $O(\log |w| + f(|z'|))$;*

FO-D-uniform *if $(L_D(C), \pi_4)$ is in* para-FO.

In the following, we sometimes leave the parameter function π_4 implicit and e.g. simply state that linear-BD-uniformity is defined as $L_{BD}(C)$ being in $\mathsf{DTIME}(n + f(k))$.

For parameterized circuit families of size $O(f(k)n^c)$, linear-BD-uniformity implies logtime-D-uniformity, which in turn implies FO-D-uniformity. The former can be shown as follows.

▶ **Lemma 18.** *Let C be a parameterized circuit family of size bounded by $f(k)n^c$. If C is linear-BD-uniform then C is logtime-D-uniform.*

Proof sketch. Let M be a machine deciding $(L_{BD}(C), \pi_4)$ in time $O(n + f(k))$, and consider the following random-access Turing machine M': it rejects all inputs not of the form $\langle G, a, p, z, z' \rangle$ with $|G|, |a|, |p|$ below $O(\log |z| + f(|z'|))$. Otherwise, it writes $\langle G, a, p, |z|, |z'| \rangle$ on a work tape and then acts as M. Now M' witnesses $(L_D(C), \pi_4) \in \mathsf{DTIME}^{\mathsf{R}}(\log n + g(k))$. A full proof can be found in the full version (Appendix B). ◀

Next we prove that logtime-D-uniformity implies FO-D-uniformity. It suffices to show the inclusion $\mathsf{DTIME}^{\mathsf{R}}(\log n + f(k)) \subseteq$ para-FO for all computable f. We do this by first showing that $\mathsf{DTIME}^{\mathsf{R}}(\log n + f(k))$ is contained in linear-BD-uniform para-AC^0, because this will be used again in Section 4.2. Through Theorems 11 and 12, the following result (and proof) can be seen as a parameterized variant of $\mathsf{DTIME}^{\mathsf{R}}(\log n) \subseteq \mathsf{FO}$ shown in [11, Proposition 7.1].

▶ **Lemma 19.** $\mathsf{DTIME}^{\mathsf{R}}(\log n + f(k))$ *is contained in linear-BD-uniform* para-AC^0*, for all computable f.*

Proof sketch. A random-access machine M can make at most $O(\log n + f(k))$ queries in $O(\log n + f(k))$ time, so a constant depth circuit family of size $2^{O(\log n + f(k))} = O(g(k)n^c)$ can nondeterministically branch over all series of query responses, verify which is correct (checking all queries in parallel), and propagate the output of a simulation of M on the correctly guessed responses to the output gate. This can be done linear-BD-uniformly because simulating an $O(\log n + f(k))$ time random-access Turing machine by answering queries from a given list of guesses takes $O(\log n + h(k))$ time on a non-random-access Turing machine.

A full proof can be found in the full version (Appendix B). ◀

We are now ready to prove that logtime-D-uniformity implies FO-D-uniformity. Here we can make use of the work of Barrington et al. [11] and that of Chen and Flum [3].

▶ **Lemma 20.** *Let C be a parameterized circuit family of size bounded by $f(k)n^c$. If C is logtime-D-uniform then C is* FO*-D-uniform.*

Proof. Assume C is logtime-D-uniform. Then, by definition, there is some computable g so that $(L_D(C), \pi_4)$ is in $\mathsf{DTIME}^{\mathsf{R}}(\log n + g(k))$. From the inclusions

$$\begin{aligned}
\mathsf{DTIME}^{\mathsf{R}}(\log n + g(k)) &\subseteq \text{linear-BD-uniform para-}\mathsf{AC}^0 && \text{by Lemma 19}\\
&= \text{Para(linear-BD-uniform } \mathsf{AC}^0) && \text{by Theorem 14}\\
&= \text{Para}(\mathsf{FO}) && \text{by Theorems 11 and 12}\\
&= \text{para-}\mathsf{FO} && \text{by definition}
\end{aligned}$$

it follows that $(L_D(C), \pi_4)$ is in para-FO, meaning that C is FO-D-uniform. ◀

4 Uniform para-AC^0 and para-$\mathsf{AC}^{0\uparrow}$

In the previous section we have seen that, for parameterized circuit families of size $f(k)n^c$, linear-BD-uniformity implies logtime-D-uniformity, which in turn implies FO-D-uniformity. It is unknown whether any of these the implications can be reversed. (See also Example 10.)

We can however show implications up-to similar-size circuit equivalence in the reverse direction, by which we mean that the uniformity conditions do not result in different complexity classes for both para-AC^0 and para-$\mathsf{AC}^{0\uparrow}$. That is, for every FO-D-uniform $O(f(k)n^c)$-sized circuit family of depth $O(d(k))$, there is a linear-BD-uniform $O(g(k)n^b)$-sized family of depth $O(d(k))$ that decides the same parameterized problem.

For para-AC^0, the equality of logtime-D-uniform para-AC^0 and FO-D-uniform para-AC^0 mirrors Theorem 11, and could be proved using the constant-depth techniques in [11]. This does not work for the $f(k)$ depth circuits of para-$\mathsf{AC}^{0\uparrow}$ (the circuits blow up). We address this by constructing logtime-uniform circuits of $f(k)$ layers that simulate FO-uniform circuits by computing their connection language. A similar layered approach can be found in [10], where they simulate $\mathsf{FO}[t(n)]$ to prove uniformity equivalences for $\mathsf{AC}[t(n)]$.

▶ **Theorem 21.** *The following classes are equal:*
1. *linear-BD-uniform* para-$\mathsf{AC}^{0\uparrow}$*,*
2. *logtime-D-uniform* para-$\mathsf{AC}^{0\uparrow}$*, and*
3. FO*-D-uniform* para-$\mathsf{AC}^{0\uparrow}$*.*

The same holds for para-AC^0*.*

Proof outline. We will use the following structure, as visualized in Figure 2 for para-AC^0. In Section 3, we have already proved that

$$\text{linear-BD-uniformity} \xRightarrow{\text{Lemma 18}} \text{logtime-D-uniformity} \xRightarrow{\text{Lemma 20}} \text{FO-D-uniform}$$

for circuit families of size $O(f(k)n^c)$, which directly gives the inclusions (1) $\subseteq$ (2) $\subseteq$ (3) for both para-AC^0 and para-$\mathsf{AC}^{0\uparrow}$. The next step is to prove (2) $\subseteq$ (1), in particular that linear-BD-uniform para-AC^0 is equal to logtime-D-uniform para-AC^0 (Lemma 24). We then use this to show (3) $\subseteq$ (1) for both para-AC^0 and para-$\mathsf{AC}^{0\uparrow}$ (Lemma 25).

The new inclusions will be proved by simulation. We will show that there are linear-BD-uniform constant-depth circuit families for deciding the connection language $L_D(C)$ of logtime-D-uniform and FO-D-uniform circuit families C of size $O(f(k)n^c)$. We then compose them into larger linear-BD-uniform circuit families to simulate C itself.

To prove that these compositions are indeed linear-BD-uniform, we first show that the class of linear-BD-uniform circuit families is in some sense closed under substitution.

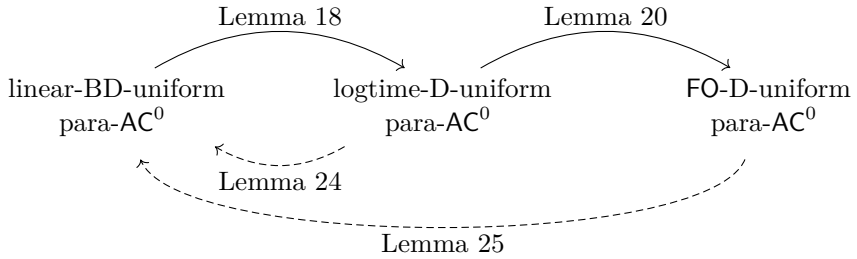


Figure 2 Visualization of the steps in the proof of Theorem 21 for para-AC^0. Solid arrows are direct implications on the level of circuit families, dashed arrows indicate implications up-to similar-size circuit equivalence. Lemma 24 is used to prove Lemma 25. The proof for para-AC$^{0\uparrow}$ follows the same structure.

4.1 Substitution

A main ingredient of our construction is that within linear-BD-uniformity we are able to *compose* circuit families. That is, given uniform circuit families A and B, we can uniformly replace some of the gates in A by circuits from B. This can be formalized as follows.

▶ **Definition 22.** *Let A, B be parameterized circuit families, let $M(G,n,k) : \mathbb{N}^3 \to 2$ be a function marking internal gates in $A_{n,k}$, and let $\mathrm{In}(G,n,k)$ be the fan-in of gate G in $A_{n,k}$. The* **substitution** *$A[B/M]$ is the parameterized circuit family where $A[B/M]_{n,k}$ consists of subcircuits $C(G)$ for each gate G in $A_{n,k}$ so that*

1. *$C(G) = G$ if $M(G,n,k) = 0$,*
2. *$C(G) = B_{\mathrm{In}(G,n,k),k}$ if $M(G,n,k) = 1$ (input and output gates become unary* OR *gates),*
3. *and if H is the ith input of G in $A_{n,k}$, then $C(H)$ is the ith input of $C(G)$ in $A[B/M]_{n,k}$.*

That is, $A[B/M]_{n,k}$ is obtained from A by replacing all marked gates with circuits of B.

It can be shown that $A[B/M]$ is linear-BD-uniform if both A and B are, and in addition M and In are reasonably computable.

▶ **Lemma 23.** *Let A, B be linear-BD-uniform parameterized circuit families of size $O(f(k)n^c)$ for some computable f. Let $M(G,n,k) : \mathbb{N}^3 \to 2$ be a function marking internal gates in $A_{n,k}$, and let $\mathrm{In}(G,n,k)$ be the fan-in of gate G in $A_{n,k}$.*

Now $A[B/M]$ is linear-BD-uniform, provided that there is a computable h such that:

(1) *$M(G,n,k)$ is computable in time $|G| + |n| + h(k)$, and*

(2) *if $M(G,n,k) = 1$ then $\mathrm{In}(G,n,k)$ is computable in time $|G| + |n| + h(k)$.*

Proof sketch. It suffices to define an admissible numbering for $D = A[B/M]$, and show that, under this numbering, $(L_{BD}(D), \pi_4)$ is in $\mathsf{DTIME}(n + f(k))$ for some computable f.

Give unmarked internal gates G in $A_{n,k}$ number $\langle 0, G\rangle$ in $D_{n,k}$. If G is a marked gate and G' is a gate in the circuit $B_{\mathrm{In}(G,n,k),k}$ replacing it, number it $\langle G, G'\rangle$ in $D_{n,k}$. This numbering based on admissible numberings of A and B leads to an admissible numbering of D (cf. Definition 5). An $O(n + f(k))$-time algorithm deciding $(L_{BD}(D), \pi_4)$ is then obtained from M, In, and $O(n + g(k))$-time algorithms for $L_{BD}(A)$ and $L_{BD}(B)$ witnessing the linear-BD-uniformity of A and B. A full proof can be found in the full version (Appendix B). ◀

We do not know whether the logtime-D-uniform (parameterized) circuit families are closed under substitution, for reasonable assumptions on M and In. It appears that two random-access Turing machines P, Q are not easily composed without a runtime overhead. For example, computing $P(x_r)$ on input $x = \langle x_l, x_r\rangle$ seems to require offsetting each query

that P makes by an $O(|x_l|)$ offset which, if done naively, adds an $O(\log |x_l|)$ overhead to every query. These naive combinations of two logtime random-access Turing machines can result in $\Omega((\log n)^2)$-time random-access Turing machines, instead of logtime.

4.2 Simulation

We are now ready to prove Theorem 21 according to the outline given at the start of the section. The first step is the indirect inclusion from logtime-D-uniformity to linear-BD-uniformity.

▶ **Lemma 24.** *Let C be a parameterized circuit family of size $O(f(k)n^c)$ and depth bounded by $d(k)$, with f, d computable. If C is logtime-D-uniform, there is an equivalent linear-BD-uniform family D of size $O(f'(k)n^{c'})$ and depth $O(d(k))$, with f' computable.*

Proof. Let a be a constant and h a computable function so that for all n, k we have that $N(n,k) = 2^{a \log n + h(k)}$ is greater than any gate number in $C_{n,k}$ (cf. Definition 5). Then the binary representation of $N(n,k)$ is computable in $O(\log n + g(k))$ time without random access for some computable g.

We describe a linear-BD-uniform circuit family D equivalent with C. Circuit $D_{n,k}$ consists of its input gates, $d(k)$ layers of $N(n,k)$ subcircuits we call **simgates**, and one final subcircuit that forwards the correct simgate to the output gate. Every simgate of layer $m{+}1$ takes all simgates of layer m as input, together with the input gates. We number the qth simgate in layer m with $\langle m, q\rangle$, and the idea is that simgate $\langle m, q\rangle$ in $D_{n,k}$ approaches (monotonically) gate q in $C_{n,k}$, so that, by layer $d(k)$, simgate $\langle d(k), q\rangle$ correctly simulates gate q in $C_{n,k}$.

See Figure 3 for the internals of a simgate. In order to approach gate q of $C_{n,k}$, simgate $\langle m, q\rangle$ queries $L_D(C)$ to select the simgates $\langle m{-}1, p\rangle$ where p feeds into q in $C_{n,k}$, and the input gates m where input gate m feeds into q in $C_{n,k}$. It also queries $L_D(C)$ to determine whether q is an AND, OR, or NOT gate, and simulates that action on its selected inputs. (By our construction in Figure 3, if $q < N(n,k)$ is not a gate in $C_{n,k}$ then simgate $\langle m, q\rangle$ outputs 0, but it has no influence on the output of $D_{n,k}$, because it is never selected.)

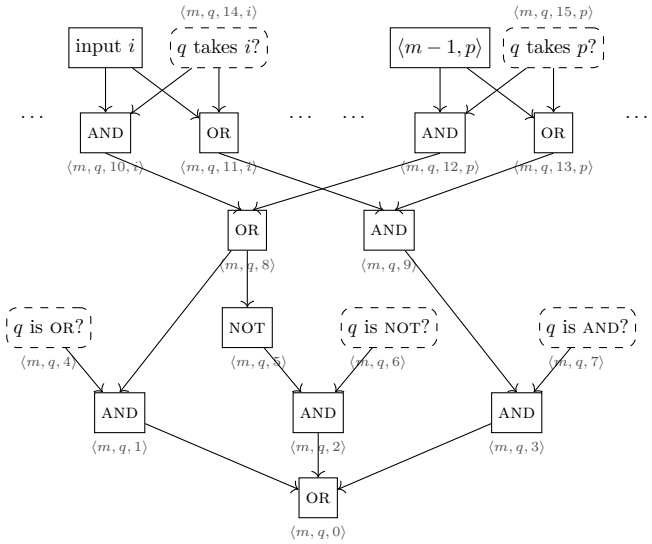


Figure 3 The internals of simgate $\langle m, q\rangle$ simulating gate q of $C_{n,k}$, annotated with an admissible numbering. It takes the n input gates and the $N(n,k)$ simgates of the previous layer as input. The boxes with rounded corners represent subcircuits that determine how to simulate q by querying $L_D(C)$. If there is no gate numbered q in $C_{n,k}$, then the output of the simgate is 0.

Assume for now that components of simgates can directly answer membership queries for $L_D(C)$. In this case the construction shows that they select their input as described above, and perform the action of q (if q is a gate in $C_{n,k}$). Then by induction, for every gate q in $C_{n,k}$, the simgate $\langle d(k), q\rangle$ in the final layer will have the same output as q.

To see this, first observe that by our assumption, the output of $\langle m+1, q\rangle$ is the same as that of q if for all internal gates p that feed into q, the output of $\langle m, p\rangle$ is equal to that of p. Define the **level** of a gate to be the length of the longest path from it to an input gate. The output of simgates $\langle m, q\rangle$ is the same as that of q if the level of q is below m. This can be shown by induction based on the following: (1) If q has level 1 in $C_{n,k}$, then q only takes input gates, and by our observation simgates $\langle m, q\rangle$ have the same output as q for all $m \geq 1$. (2) If q has level $m+1$ and simgates $\langle m', p\rangle$ have the same output as p for all p of lower level and $m' \geq m$, then by our observation $\langle m'', q\rangle$ has the same output as q for all $m'' \geq m+1$.

Finally, to make $D_{n,k}$ behave as desired, it should feed into the output gate the output of simgate $\langle d(k), q\rangle$ for the gate q that feeds into the output gate in $C_{n,k}$. This is the role of the final subcircuit mentioned at the start of the proof. It takes all simgates in the final layer as input, and queries $L_D(C)$ to propagate the output of the correct $\langle d(k), q\rangle$.

The circuits $D_{n,k}$ have very regular structure. Each $D_{n,k}$ is essentially a matrix of $d(k)$ layers of $N(n,k)$ simgates, and each simgate gets all the simgates of the previous layer as input together with the input gates. Apart from the rounded query blocks, the internals of the simgates are also easily seen to be linear-BD-uniform, using for example the admissible numbering in Figure 3. It remains to show how the queries in the simgates are implemented.

Simgate queries. The $L_D(C)$ queries represented by the dashed query blocks in Figure 3 can be implemented by plugging in linear-BD-uniform circuits for $L_D(C)$. A constant-depth linear-BD-uniform circuit family E deciding $L_D(C)$ exists by Lemma 19, and its circuits can be substituted in linear-BD-uniformly by Lemma 23. It is left to prove that the circuits can be given the appropriate query inputs in a linear-BD-uniform way.

For now, we work with the circuit family D' in which the circuits of E have not yet been substituted. Here gates $\langle m, q, 4\rangle$ (cf. Figure 3) are just AND gates that will be replaced with circuits from E later to obtain D from D'. We add constant circuits V_0 and V_1 to every $D'_{n,k}$, where V_0 has the constant output 0, and V_1 is constant 1. The circuit of E replacing $\langle m, q, 4\rangle$ should decide whether gate q in $C_{n,k}$ is an OR gate. This can be achieved by giving gate $\langle m, q, 4\rangle$ the input $I = \langle q, 1, \epsilon, 1^n, 1^k\rangle$. That is, in terms of connection languages, we want $w = \langle\langle m, q, 4\rangle, V_b, i, n, k\rangle \in L_{BD}(D')$ if and only if the ith bit of I is b. It can be verified that, given such w, the ith bit of I can be decided in time $|w| + g(k)$ for some computable g, and so these queries can be answered in time $|w| + g(k)$ as required for linear-BD-uniformity.

This addresses the type-queries in the simgates. The other queries, those asking whether (input) gate p feeds into q in $C_{n,k}$, are handled similarly, with a small complication: checking whether p feeds into q can be reduced to checking whether p is the ith input of q for any $i \leq N(n,k)$, so the dashed blocks in Figure 3 corresponding to this type of query are actually OR gates over $N(n,k)$ gates that will be substituted by circuits of E, where the ith gate will be given an $L_D(C)$-query to determine whether p is the ith input of q.

In summary: the queries in the simgates are implemented by hard-wiring $L_D(C)$-queries into the query blocks in a linear-BD-uniform manner, and then substituting in a linear-BD-uniform circuit family E that decides $L_D(C)$ to answer them. The resulting circuit family D is linear-BD-uniform by Lemma 23, and is equivalent with C by design. ◀

Theorem 12 (mentioned in Section 2.5) can be proved similarly. Using simgates, we can now also prove the following.

▶ **Lemma 25.** *Let C be a parameterized circuit family of size $O(f(k)n^c)$ and depth bounded by $d(k)$, with f, d computable. If C is* FO*-D-uniform, there is an equivalent linear-BD-uniform family D of size $O(f'(k)n^{c'})$ and depth $O(d(k))$, with f' computable.*

Proof. By the FO-D-uniformity of C, we have $(L_D(C), \pi_4) \in \text{Para}(\mathsf{FO})$. Theorem 11 gives that $(L_D(C), \pi_4)$ is in Para(logtime-D-uniform AC^0), which by Theorem 14 is equal to logtime-D-uniform para-AC^0. Hence, by Lemma 24, $(L_D(C), \pi_4)$ is in linear-BD-uniform para-AC^0. The rest of the proof is now the same as for Lemma 24, using a simgate-construction of $d(k)$ layers, and answering $L_D(C)$-queries in the simgates by substituting in a family witnessing that $(L_D(C), \pi_4)$ is in linear-BD-uniform para-AC^0. ◀

We have now proved our main result.

Proof of Theorem 21. For both para-AC^0 and para-$\mathsf{AC}^{0\uparrow}$, Item (1) implies Item (2) by Lemma 18, and (2) implies (3) by Lemma 20. Finally, Item (3) implies Item (1) by Lemma 25. (For para-AC^0, the last implication is obtained from Lemma 25 by setting $d(k) = O(1)$.) ◀

We can also observe the following, which suggests that FO-D-uniformity is not too coarse for para-$\mathsf{AC}^{0\uparrow}$. (This mirrors that (FO-uniform AC^0)-uniform AC^0 equals FO-uniform AC^0.)

▶ **Corollary 26.** *(*FO*-D-uniform* para-$\mathsf{AC}^{0\uparrow}$*)-uniform* para-$\mathsf{AC}^{0\uparrow}$ = FO*-D-uniform* para-$\mathsf{AC}^{0\uparrow}$.

Proof sketch. If $(L_D(C), \pi_4)$ is in FO-uniform para-$\mathsf{AC}^{0\uparrow}$, then by Theorem 21 it lies in linear-BD-uniform para-$\mathsf{AC}^{0\uparrow}$. Follow the proof of Lemma 24, now using linear-BD-uniform para-$\mathsf{AC}^{0\uparrow}$-families for the query blocks. ◀

5 Extended uniformity

Thus far we considered direct uniformity notions, where the connection language only contains information on directly related gates. Extended uniformity notions, where the connection language also describes paths between gates, are of interest because logtime-E-uniform circuit classes such as logtime-E-uniform AC^0 and NC^i are equal to natural alternating Turing machine classes [16]. It is unknown whether logtime-D-uniformity and logtime-E-uniformity are equivalent in general, but it has been shown that logtime-D-uniform AC^0 is equal to logtime-E-uniform AC^0 [16, Theorem 4.31]. We will show that this also holds for para-AC^0 and, more interestingly, for para-$\mathsf{AC}^{0\uparrow}$.

▶ **Definition 27.** *Let C be a circuit. A* **path** *from gate G to gate G' is a list of steps $p = \langle p_1, p_2, \ldots, p_m \rangle$ such that there exist gates $G_0, \ldots, G_m, G_{m+1}$ in C where for each i, G_{i+1} feeds into the p_{i+1}th input wire of G_i, with $G_0 = G$ and $G_{m+1} = G'$.*

In the extended connection language, the p in $\langle G, a, p, z, z' \rangle$ are paths from G to a [16].

▶ **Definition 28.** *Let C be a parameterized circuit family. The* **extended connection language** *$L_E(C)$ consists of all strings of the form $\langle G, a, p, z, z' \rangle$ where, writing n for $|z|$ and k for z':*

1. *G is a gate in $C_{n,k}$;*
2. *if $p = \epsilon$ then $a < 3$ indicates the type of G (one of* AND, OR, NOT*);*
3. *otherwise, p is a path from G to a.*

In [16, p. 123], Vollmer defines logtime-E-uniformity for non-parameterized polynomial-size constant-depth circuit families C as $L_E(C)$ being in $\mathsf{DTIME}^{\mathsf{R}}(\log n)$. We adapt this to $O(d(k))$-depth parameterized circuit families as follows.

▶ **Definition 29.** *A parameterized circuit family of size $O(f(k)n^c)$ and depth $O(f(k))$ is* **logtime-E-uniform** *if $(L_E(C), \pi_4)$ is $\mathsf{DTIME}^{\mathsf{R}}(\log n + g(k))$ for a computable g.*

For para-$\mathsf{AC}^{0\uparrow}$, this definition is quite strict: there are logtime-D-uniform circuit families of size $O(f(k)n^c)$ and depth $d(k)$ that contain valid paths p of $d(k)$ steps, where every step has size $\Omega(\log n)$. The encoding of such paths p has length $\Omega(d(k)\log n)$. No $\mathsf{DTIME}^{\mathsf{R}}(\log n+g(k))$ machine in Definition 28 has time to read the complete path p. Nevertheless, these circuit families still have logtime-E-uniform representatives for para-$\mathsf{AC}^{0\uparrow}$.

▶ **Lemma 30.** *Let A be a parameterized circuit family of size $O(f(k)n^c)$ and depth bounded by $d(k)$, with f, d computable. If A is linear-BD-uniform, there is an equivalent logtime-E-uniform circuit family C of size $O(f'(k)n^{c'})$ and depth $O(d(k))$, with f' computable.*

Proof sketch. As in the proofs of Lemmas 24 and 25, C will be made up of $d(k)$ layers of simgates. The main idea is that only steps between layers are difficult, and that in order to decide to which simgate a path leads, we only need to read the last inter-layer step. We achieve this by making the number of simgates per layer exactly $2^{L(n,k)} - 1$ for some appropriate $L(n, k)$. Then, if C were to consist only of these simgates, and the simgates were actual gates instead of subcircuits, it could be checked whether a path p moves from simgate $\langle m, q\rangle$ to $\langle m', q'\rangle$ by making sure that

1. each step in p has length at most $L(n, k)$,
2. $d(k) \geq m \geq m' \geq 1$ and p contains $m - m'$ steps, and
3. the last step in p is q'.

So, only the final step in p has to be read. For the others it suffices to read their lengths. From this it can be shown that the whole procedure can be performed in $O(h(k) + \log n)$ time for some computable h. The full version contains a full proof in Appendix B. ◀

Because logtime-E-uniform families of size $O(f(k)n^c)$ and depth $f(k)$ are by definition also logtime-D-uniform and hence (by Theorem 21) equivalent to a linear-BD-uniform family of similar dimensions, we may now conclude the following.

▶ **Corollary 31.** *Logtime-E-uniform* para-$\mathsf{AC}^{0\uparrow}$ *equals logtime-D-uniform* para-$\mathsf{AC}^{0\uparrow}$*, and logtime-E-uniform* para-AC^0 *equals logtime-D-uniform* para-AC^0*.*

We can also define FO-E-uniformity as $(L_E(C), \pi_4)$ being para-FO. By the same proof as Lemma 20, it can then be shown that logtime-E-uniformity implies FO-E-uniformity for parameterized circuit families of size $O(f(k)n^c)$, and thus (because FO-E-uniformity implies FO-D-uniformity, and by then applying Theorem 21) FO-E-uniform para-$\mathsf{AC}^{0\uparrow}$ is equivalent with logtime-D-uniform para-$\mathsf{AC}^{0\uparrow}$. The same also holds for para-AC^0.

6 Relation to descriptive complexity

In this section we discuss a descriptive classification of logtime-uniform para-$\mathsf{AC}^{0\uparrow}$ mirroring the equivalence of FO and logtime-uniform AC^0 in Theorem 11.

For non-parameterized circuits, Barrington and Immerman [10] prove that for sufficiently constructible polynomially bounded functions t, logtime-D-uniform and FO-D-uniform $\mathsf{AC}[t(n)]$ are equal to the class $\mathsf{FO}[t(n)]$ of languages decided by iterated first order sentences.

A similar observation can be made in the parameterized setting. Define para-$\mathsf{AC}[f(k)]$ in terms of parameterized circuit families of size $O(f(k)n^c)$ and depth $O(f(k))$, so that para-$\mathsf{AC}^{0\uparrow}$ is the union of para-$\mathsf{AC}[f(k)]$ over the computable functions f. Based on the definition of $\mathsf{FO}[t(n)]$ in [9, Definition 4.24], we define para-$\mathsf{FO}[f(k)]$ as follows.

Write $(\forall x.\phi)\psi$ for $(\forall x)(\phi(x) \to \psi)$ and $(\exists x.\phi)\psi$ for $(\exists x)(\phi(x) \wedge \psi)$. A **quantifier block** is a partial formula of the form $(Q_0 x_0.M_0)(Q_1 x_1.M_1)\dots(Q_m x_m.M_m)$, where the Q_i are quantifiers, the x_i are (not necessarily distinct) variable letters, and the M_i are quantifier-free formulas in the language of FO.

▶ **Definition 32.** *A parameterized problem (Q, κ) is in* para-FO$[f(k)]$ *if there is a computable g, a quantifier block M, a quantifier-free formula ψ, and a tuple of constants c such that*

$$y \models M^{f(\kappa(x))}\psi(c) \iff y \in \{\langle x, g(\kappa(x))\rangle : x \in Q\}$$

where $M^{f(\kappa(x))}\psi(c)$ stands for the sentence obtained by concatenating $f(\kappa(x))$ copies of M with ψ, and then substituting c_i for free occurrences of variables x_i.

Intuitively, a parameterized problem (Q, κ) is in para-FO$[f(k)]$ if it is first-order describable in $|x|, f(\kappa(x))$ and iterations of length $f(\kappa(x))$, that is, if $x \in Q \iff x \models M^{f(\kappa(x))}\psi(c)$ holds. That this corresponds with Definition 32 can be made precise by adding constant symbols for $|x|$ and $f(\kappa(x))$ to the language, and extending the domain of x from $|x|$ to $\max(|x|, f(\kappa(x)))$.

Following the proof strategies in [10] (see also [9]) it follow that, writing C for the computable functions $\mathbb{N} \to \mathbb{N}$:

$$\text{FO-D-uniform para-AC}^{0\uparrow} = \bigcup_{f \in \mathsf{C}} \text{FO-uniform para-AC}[f(k)] = \bigcup_{f \in \mathsf{C}} \text{para-FO}[f(k)],$$

thus linking logtime-D-uniform para-AC$^{0\uparrow}$ to a descriptive complexity class. Following [10, 9] further leads to an alternative, descriptive complexity-based proof for Theorem 21.

7 Conclusion

We have shown that, for para-AC^0 and para-AC$^{0\uparrow}$, various uniformity conditions found in ordinary circuit complexity are again equivalent in the sense that the respective induced complexity classes are equal. This can be used to repair uniformity gaps in the literature.

For example, while our result does not directly demonstrate that the circuit family mentioned in [1, Theorem 3.2] is logtime-D-uniform as claimed, it does prove that there is a suitable logtime-D-uniform circuit family equivalent with it. Since the circuits are described in terms of additions, multiplications, and modulo operations of numbers below a polynomial in n and parameter k, it is straightforward to show that the circuit family is FO-D-uniform, and so, by Theorem 21, there is an equivalent logtime-D-uniform family.

We obtain our equivalences via the stricter parameterized linear-uniformity condition, proving a substitution lemma and then explicitly constructing layered linear-uniform families for given FO-uniform families. In a sense this can be seen as a direct, circuit-oriented variant of the approach in [10]. Here they prove the equivalence of (direct, non-parameterized) uniformity conditions for depth-$t(n)$ circuits via the descriptive complexity class FO$[t(n)]$, constructing layered logtime-uniform circuits deciding FO$[t(n)]$-formulas.

Our results show the equivalence between the uniformity conditions not on the circuit family level but only on the complexity class level. This suffices for theoretical applications like [1, Theorem 3.2] mentioned above, but is not fine-grained enough for questions of work efficiency. For example, the proof of Theorem 21 gives for an FO-D-uniform circuit family of size $f(k)n^c$ a logtime-D-uniform circuit family of size greater than $f(k)n^{3c+1}$. This gap limits the study of open problems in fine-grained parallel complexity. It would therefore be interesting to study the inherent cost of the conversion, and to also consider which uniformity condition is the most natural one from a (parameterized) parallel complexity perspective.

References

1. Max Bannach, Christoph Stockhusen, and Till Tantau. Fast Parallel Fixed-parameter Algorithms via Color Coding. In *Leibniz International Proceedings in Informatics*, IPEC '15, page 12, 2015. Extended version at arXiv:1509.06984. doi:10.4230/LIPICS.IPEC.2015.224.
2. Max Bannach and Till Tantau. Parallel Multivariate Meta-Theorems. In *Leibniz International Proceedings in Informatics*, IPEC '16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.IPEC.2016.4.
3. Yijia Chen and Jörg Flum. Some lower bounds in parameterized AC0. *Information and Computation*, 267:116–134, 2019. doi:10.1016/j.ic.2019.03.008.
4. Yijia Chen, Jörg Flum, and Xuangui Huang. Slicewise Definability in First-Order Logic with Bounded Quantifier Rank. In *26th EACSL Annual Conference on Computer Science Logic (CSL 2017)*, pages 19:1–19:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.CSL.2017.19.
5. Michael Elberfeld, Christoph Stockhusen, and Till Tantau. On the Space and Circuit Complexity of Parameterized Problems: Classes and Completeness. *Algorithmica*, 71(3):661–701, 2015. doi:10.1007/s00453-014-9944-y.
6. Jörg Flum and Martin Grohe. Describing parameterized complexity classes. *Information and Computation*, 187(2):291–319, 2003. doi:10.1016/S0890-5401(03)00161-5.
7. Jörg Flum and Martin Grohe. *Parameterized complexity theory*. Texts in theoretical computer science an EATCS series. Springer, Berlin Heidelberg, 2006. doi:10.1007/3-540-29953-X.
8. David Harvey and Joris van der Hoeven. Integer multiplication in time $O(n \log n)$. *Annals of Mathematics*, 193(2):563–617, 2021. doi:10.4007/annals.2021.193.2.4.
9. Neil Immerman. *Descriptive Complexity*. Springer New York, New York, NY, 1999. doi:10.1007/978-1-4612-0539-5.
10. David A. Mix Barrington and Neil Immerman. Time, hardware, and uniformity. In *Proceedings of IEEE 9th Annual Conference on Structure in Complexity Theory*, pages 176–185. IEEE Comput. Soc. Press, 1994. doi:10.1109/SCT.1994.315806.
11. David A. Mix Barrington, Neil Immerman, and Howard Straubing. On uniformity within NC^1. *Journal of Computer and System Sciences*, 41(3):274–306, 1990. doi:10.1016/0022-0000(90)90022-D.
12. Michał Pilipczuk, Sebastian Siebertz, and Szymon Toruńczyk. Parameterized circuit complexity of model-checking on sparse structures. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 789–798, 2018. doi:10.1145/3209108.3209136.
13. Kenneth W. Regan and Heribert Vollmer. Gap-languages and log-time complexity classes. *Theoretical Computer Science*, 188(1):101–116, 1997. doi:10.1016/S0304-3975(96)00288-5.
14. Walter L. Ruzzo. On uniform circuit complexity. *Journal of Computer and System Sciences*, 22(3):365–383, 1981. doi:10.1016/0022-0000(81)90038-6.
15. Rahul Santhanam and Ryan Williams. On Uniformity and Circuit Lower Bounds. *computational complexity*, 23(2):177–205, 2014. doi:10.1007/s00037-014-0087-y.
16. Heribert Vollmer. *Introduction to Circuit Complexity: A Uniform Approach*. Texts in Theoretical Computer Science. Springer, 1999. doi:10.1007/978-3-662-03927-4.

Geodetic Set on Graphs of Constant Pathwidth and Feedback Vertex Set Number

Prafullkumar Tale
Indian Institute of Science Education and Research Pune, India

Abstract

In the Geodetic Set problem, the input consists of a graph G and a positive integer k. The goal is to determine whether there exists a subset S of vertices of size k such that every vertex in the graph is included in a shortest path between two vertices in S. Kellerhals and Koana [IPEC 2020; J. Graph Algorithms Appl 2022] proved that the problem is W[1]-hard when parameterized by the pathwidth or the feedback vertex set number of the input graph. They posed the question of whether the problem admits an XP-algorithm when parameterized by the combination of these two parameters. We answer this in the negative by proving that the problem remains NP-hard even on graphs of constant pathwidth and feedback vertex set number.

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

Keywords and phrases Geodetic Sets, NP-hardness, Constant Treewidth

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.28

Related Version *Full Version*: https://arxiv.org/abs/2504.17862

Funding Research supported by INSPIRE Faculty Fellowship offered by Department of Science and Technology, Government of India.

1 Introduction

Consider a simple undirected graph G with vertex set $V(G)$ and edge set $E(G)$. For two vertices $u, v \in V(G)$, let $I(u, v)$ be the set of all vertices in G that are part of some shortest path between u and v. For a subset S of vertices, we generalize this definition to $I(S)$ by taking the union of $I(u, v)$ for all pairs of vertices u, v in S. A subset of vertices T is said to be *covered* by S if $T \subseteq I(S)$. A set of vertices S is a *geodetic set* if $V(G)$ is covered by S. In the Geodetic Set problem, the input is a graph G and a positive integer k, and the objective is to determine whether there is a geodetic set of size k. This problem was introduced in 1993 by Harary, Loukakis, and Tsouros [27]. We refer readers to [9, 21] for application of the problem. See [22, 37] for a broader discussion on geodesic convexity in graphs.

The Geodetic Set problem falls under the broad category of *metric graph problems*. These problems are defined using a metric on the graph, with the shortest distance between two vertices being the most commonly used metric. Metric graph problems include many important NP-complete graph problems such as Distance d-Dominating Set (also called (k, d)-Center), Distance d-Independent Set (also called d-Scattered Set), and Metric Dimension to name a few. These problems have played a central role in the development of both classical and parameterized algorithms and complexity theory [32, 33, 3, 35, 4, 10, 23, 29, 11], as they behave quite differently from their more "local" (neighborhood-based) counterparts such as Dominating Set or Independent Set.

Due to the non-local nature of metric graph problems, most algorithmic techniques fail to yield positive results. Consider the case of the Metric Dimension problem in which an input is a graph G and an integer k, and the objective is to determine whether there

© Prafullkumar Tale;
licensed under Creative Commons License CC-BY 4.0

is a set S of at most k vertices such that for every pair of distinct vertices u and v, there exists a vertex in S that has different distances to u and v. In a seminal paper, Hartung and Nichterlein [28] proved that the problem is W[2]-hard when parameterized by the solution size k. This motivated the structural parameterization of the problem. See, for example, Galby et al. [25] for an overview. The complexity of Metric Dimension parameterized by treewidth remained an open problem for a long time, until Bonnet and Purohit [5] made the first breakthrough by proving that the problem is W[1]-hard when parameterized by pathwidth. This result was strengthened in two directions: Li and Pilipczuk [35] proved that it is para-NP-hard when parameterized by pathwidth, whereas Galby et al. [25] showed that the problem is W[1]-hard when parameterized by pathwidth plus the feedback vertex set number of the graph.

Continuing the exploration of hardness results, we mention some results that hold for both Metric Dimension and Geodetic Set. Foucaud et al. [23] demonstrated that these two problems on graphs of bounded diameter admit double exponential conditional lower bounds when parameterized by treewidth, along with a matching algorithm. The same set of authors also proved that both problems admit "exotic" conditional lower bounds when parameterized by the vertex cover number [24]. Bergougnoux et al. [4] showed that enumerating minimal solution sets for both problems in split graphs is equivalent to enumerating minimal transversals of hypergraphs, arguably the most important open problem in algorithmic enumeration.

As illustrated in the examples above, Metric Dimension and Geodetic Set behave similarly concerning hardness results. Kellerhals and Koana [34] proved that Geodetic Set is W[1]-hard when parameterized by the pathwidth and feedback vertex set number (and the solution size) combined, a result similar to that of Galby et al. [25] regarding Metric Dimension. However, a counterpart to the result of Li and Pilipczuk [35] for Geodetic Set is not yet known. More formally, we do not know the complexity of the Geodetic Set problem on graphs of constant treewidth. Chakraborty et al. [8] mentioned the possibility of obtaining XP-algorithm parameterized by the treewidth. Kellerhals and Koana [34] posed an even stronger question: Does Geodetic Set admit an XP-algorithm when parameterized by the pathwidth and feedback vertex set number? We answer these questions in the negative.

▶ **Theorem 1.** *Geodetic Set is* NP*-complete even on graphs of pathwidth at most* 17 *and feedback vertex set number at most* 13*.*

This result is particularly surprising in light of the closely related problem Geodetic Hull. In Geodetic Hull, the input is a graph G and an integer k, and the objective is to determine whether there is a vertex set S of size k such that $I^j[S] = V(G)$ for some $j > 0$, where $I^0[S] = S$ and $I^j[S] = I[I^{j-1}[S]]$ for $j > 0$. It is easy to see that both these problems are trivial on trees since selecting all leaves is both necessary and sufficient for a solution. Kante et al. [30] proved, among other things, that the problem admits an XP-algorithm parameterized by treewidth. Hence, while Geodetic Hull is polynomial-time solvable on graphs of constant treewidth, Geodetic Set remains NP-hard.

Related Work. As is often the case with metric-based problems, Geodetic Set is computationally hard – even on highly structured graph classes. See [2, 7, 8, 9, 12, 13, 18, 19, 20, 21, 27] for various earlier hardness results. Geodetic Set can be solved in polynomial time on split graphs [19, 20], well-partitioned chordal graphs [1], outerplanar graphs [36], ptolemaic graphs [22], cographs [19], distance-hereditary graphs [31], block-cactus graphs [21], solid grid graphs [8, 12], and proper interval graphs [21]. A two-player game variant of Geodetic Set was introduced in [6].

As mentioned before, Kellerhals and Koana in [34] studied the parameterized complexity of Geodetic Set. They observed that a reduction from [19] implies that the problem is W[2]-hard when parameterized by the solution size – even for chordal bipartite graphs. They proved the problem to be W[1]-hard for the parameters solution size, feedback vertex set number, and pathwidth, combined [34]. On the positive side, they showed that Geodetic Set is FPT for the parameters treedepth, modular-width (more generally, clique-width plus diameter), and feedback edge set number [34]. Chakraborty et al. [8] proved that the problem is FPT on chordal graphs when parameterized by the treewidth. On the approximation side, it is known that the minimization variant of the problem is NP-hard to approximate within a factor of $o(\log n)$, even on graphs of diameter 2 [9] and subcubic bipartite graphs of arbitrarily large girth [15].

2 Preliminaries

For an integer n, we let $[n] = \{1, \ldots, n\}$. We use standard graph-theoretic notation and refer the reader to [16] for any undefined notation. For an undirected graph G, the sets $V(G)$ and $E(G)$ denote its set of vertices and edges, respectively. Two vertices $u, v \in V(G)$ are *adjacent* or *neighbors* if $(u, v) \in E(G)$. We say a vertex v is a *pendant vertex* if its degree is one. A vertex v is a *branching vertex* if its degree is at least three. A path is a collection of vertices $\{v_1, v_2, \ldots, v_\ell\}$ such that $(v_i, v_{i+1}) \in E(G)$ for every i in $[\ell - 1]$. In this case, v_1 and v_ℓ are two endpoints, whereas all the other vertices are called *internal vertices*. We say a path is simple if all its internal vertices have degree precisely two. The length of a path is the number of edges in it. The *distance* between two vertices u, v in G, denoted by $\mathrm{dist}(u, v)$, is the length of a shortest path connecting u to v. For a subset S of $V(G)$, we denote the graph obtained by deleting S from G by $G - S$. Recall that a subset $S \subseteq V(G)$ is a *geodetic set* if for every $u \in V(G)$ there exist $s_1, s_2 \in S$ such that u lies on a shortest path from s_1 to s_2. Consider a pendant vertex v in G. Note that v does not belong to any shortest path between any pair x, y of vertices which are distinct from v. Hence, v belongs to every geodetic set of G. This observation was also made in [13].

We refer readers to Cygan et al. [14] for unspecified terms related to parameterized complexity. We define some of the parameters mentioned in this article. For a graph G, a set $X \subseteq V(G)$ is a *feedback vertex set* of G if $G - X$ is an acyclic graph. The *feedback vertex set number* of graph G, denoted by $\mathtt{fvs}(G)$, is the size of a minimum feedback vertex set of G. We denote by $\mathtt{tw}(G)$ and $\mathtt{pw}(G)$ as the treewidth and pathwidth of graph G, respectively. It is easy to verify that $\mathtt{tw}(G) \leq \mathtt{fvs}(G) + 1$ and $\mathtt{tw}(G) \leq \mathtt{pw}(G)$ whereas $\mathtt{pw}(G)$ and $\mathtt{fvs}(G)$ are incomparable.

We define an equivalent definition of pathwidth via mixed search games. In a mixed search game, a graph G is considered a system of tunnels. Initially, all edges are contaminated by a gas. An edge is cleared by placing searchers at both its endpoints simultaneously or by sliding a searcher along the edge. A cleared edge is re-contaminated if there is a path from an uncleared edge to the cleared edge without any searchers on its vertices or edges. A search is a sequence of operations that can be of the following types: (i) placement of a new searcher on a vertex; (ii) removal of a searcher from a vertex; (iii) sliding a searcher on a vertex along an incident edge to its other endpoint and placing the searcher there. A search strategy is winning if, after its termination, all edges are cleared. The mixed search number of a graph G, denoted by $\mathtt{ms}(G)$, is the minimum number of searchers required for a winning strategy of mixed searching on G. Takahashi et al. [38] proved that $\mathtt{pw}(G) \leq \mathtt{ms}(G) \leq \mathtt{pw}(G) + 1$, which we use to bound the pathwidth of the graphs obtained in the reduction.

3 NP-Hardness

In this section, we prove that the problem is para-NP-hard when parameterized by pathwidth and the feedback vertex set number of the input graph. We present a polynomial-time reduction from the 3-Dimensional Matching problem, which is known to be NP-hard [26, SP 1]. For notational convenience, we work with the following definition of the problem. The input consists of a universe $\mathcal{U} = \{\alpha, \beta, \gamma\} \times [n]$, a family $\mathcal{S}$ of m subsets of $\mathcal{U}$ such that for every set $S \in \mathcal{S}$, $S = \{(\alpha, a), (\beta, b), (\gamma, c)\}$ for some $a, b, c \in [n]$. The goal is to find a subset $\mathcal{S}' \subseteq \mathcal{S}$ of size n that partitions $\mathcal{U}$.

Reduction

The reduction takes as input an instance $(\mathcal{U}, \mathcal{S})$ of 3-Dimensional Matching and returns an instance (G, k) of Geodetic Set in polynomial time where pathwidth plus feedback vertex set number of G is a constant. Recall that a branching vertex is a vertex of degree at least three. We start by specifying the common branching vertices in G and the lengths of simple paths connecting them.

Common Branching Vertices. The reduction starts constructing G by adding the following vertices: $\{g_1, g_2, g_3, g_4\}$, $\{p^\alpha, q^\alpha, r^\alpha\}$, $\{p^\beta, q^\beta, r^\beta\}$, and $\{p^\gamma, q^\gamma, r^\gamma\}$. All set-encoding gadgets and element-encoding gadgets, which we define below, are connected via these common branching vertices. These common branching vertices are denoted using grey shaded region in all the figures. Apart from the vertices in $\{q^\alpha, q^\beta, q^\gamma\}$, the reduction adds pendant vertices adjacent to each common branching vertex. In all the figures, vertices that are adjacent to a pendant vertex are highlighted by enclosed squares around them. Define an integer M such that $M \in \mathcal{O}(n^2)$ and for any $a \in [n]$, we have $0.99 \cdot M^2 \leq M^2 - 2aM, M^2 + 2aM \leq 1.01 \cdot M^2$. The reduction connects g_1 to g_2 and g_3 to g_4 using a path of length M^2 for each connection. See Figure 1.

Set-encoding Gadget. The reduction adds n identical set-encoding gadgets denoted by $\mathcal{S}^1, \mathcal{S}^2, \ldots, \mathcal{S}^n$. For any $i \in [n]$, set-encoding gadget $\mathcal{S}^i$ contains sets of vertices X^i, Y^i, and two auxiliary vertices z_1^i and z_2^i. Sets X^i and Y^i contain m vertices each. The reduction adds matching edges across X^i and Y^i. Each of these matching edges and their endpoints corresponds to a unique set in $\mathcal{S}$. Then, the reduction replaces each of these matching edges with a simple path of length M^2. For each vertex in Y^i, the reduction adds a pendant vertex adjacent to it. It connects z_1^i with every vertex in $\{z_2^i\} \cup X^i$ using a simple path of length M^2 for each connection.

We now specify how the vertices mentioned above are connected to common branching vertices. The reduction connects (i) g_1 to z_1^i and z_2^i, (ii) g_2 to z_1^i, and (iii) g_4 to every vertex in Y^i, using paths of length M^2 for each connection. It connects g_2 to every vertex in Y^i using a path of length $M^2 - 1$. We remark that these are the only paths that are of length $M^2 - 1$. To specify the connection to the remaining common branching vertices, consider a set $S = \{(\alpha, a'), (\beta, b'), (\gamma, c')\}$ in $\mathcal{S}$ and the vertex x corresponding to it in X^i. The reduction connects x to all the remaining common branching vertices such that

- $\text{dist}(x, p^\alpha) = M^2 + 2a'M$, $\text{dist}(x, q^\alpha) = M^2 - a'M$, and $\text{dist}(x, r^\alpha) = M^2 - 2a'M$;
- $\text{dist}(x, p^\beta) = M^2 + 2b'M$, $\text{dist}(x, q^\beta) = M^2 - b'M$, and $\text{dist}(x, r^\beta) = M^2 - 2b'M$; and
- $\text{dist}(x, p^\gamma) = M^2 + 2c'M$, $\text{dist}(x, q^\gamma) = M^2 - c'M$, and $\text{dist}(x, r^\gamma) = M^2 - 2c'M$.

See Figure 2 for an illustration.

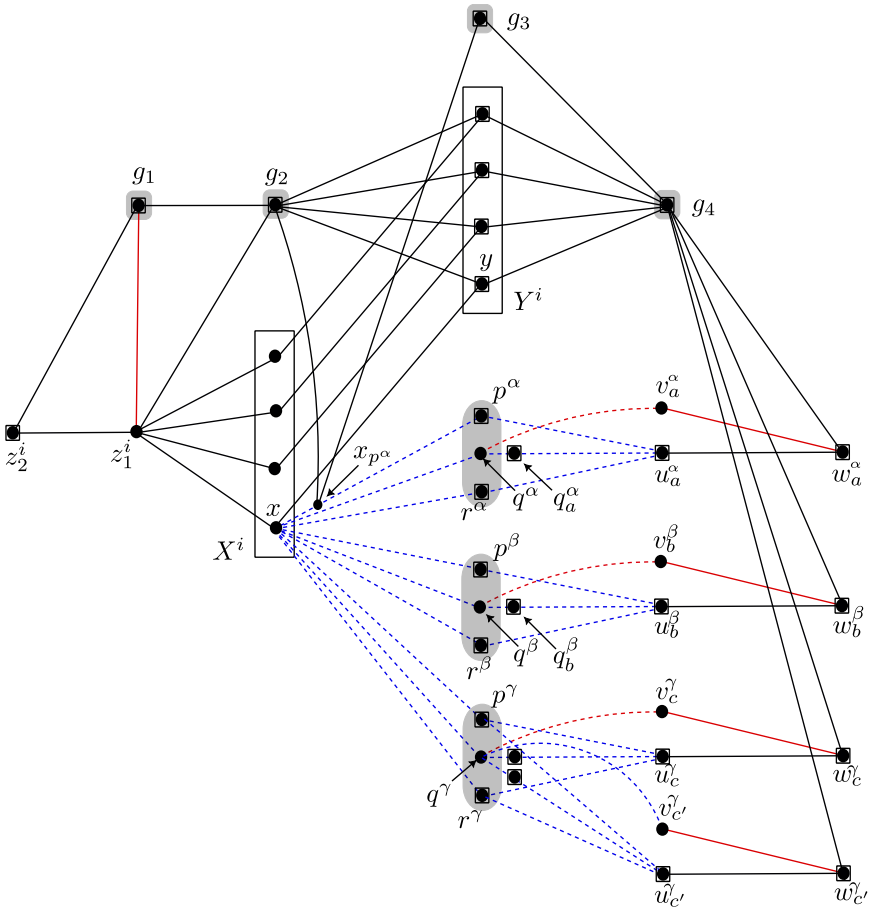


Figure 1 The figure shows the i^{th} copy of set-encoding gadget $\{z_2^i, z_1^i\} \cup X^i \cup Y^i$, element-encoding gadgets encoding elements of the form (α, a), (β, b), (γ, c) and (γ, c'), for some $a, b, c, c' \in [n]$, and the common branching vertices. Each solid (black or red) edge represents a path of fixed length which is either M^2 or $M^2 - 1$. Each dotted (blue or red) edge represents a path whose length depends on the set or element corresponding to one of its endpoints. Apart from the internal vertices in the red edges, all the other vertices are covered by pendant vertices in the graph. For clarity, we show connecting paths starting at only one vertex in X^i. Moreover, it only shows one of the nine vertices, viz x_{p^α}, that are adjacent to x and are connected to g_2 and g_3.

Now, consider the vertex on the path connecting x to p^α which is adjacent to x. We denote this vertex by x_{p^α}. The reduction connects x_{p^α} with g_2 and g_3 using the paths of length M^2 each. It repeats this process to add vertices x_{q^α}, x_{r^α}, x_{p^β}, x_{q^β}, x_{r^β}, x_{p^γ}, x_{q^γ}, and x_{r^γ}, and their connection to g_2 and g_3.

Element-Encoding Gadget. Consider an element (α, a) in $\mathcal{U}$. The reduction adds three vertices u_a^α, v_a^α and w_a^α. It adds a pendant vertex adjacent to u_a^α and another pendant vertex adjacent to w_a^α. It connects w_a^α with u_a^α and v_a^α using a simple path of length M^2 for each connection.

The reduction connects u_a^α with $\{p^\alpha, q^\alpha, r^\alpha\}$ using simple paths such that
- $\text{dist}(u_a^\alpha, p^\alpha) = M^2 - 2aM$, $\text{dist}(u_a^\alpha, q^\alpha) = M^2 + aM$, and $\text{dist}(u_a^\alpha, r^\alpha) = M^2 + 2aM$.

It connects v_a^α (only) to q^α such that $\text{dist}(v_a^\alpha, q^\alpha) = M^2 + aM$. See Figure 2 for an illustration. Consider the vertex in the simple path connecting q^α and u_a^α which is adjacent with q^α. We denote the vertex by q_a^α. The reduction adds a pendant vertex and makes it adjacent with q_a^α.

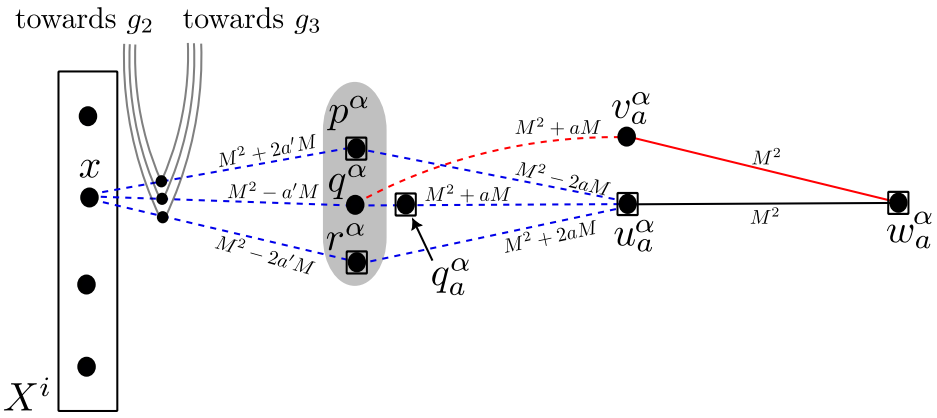


Figure 2 On the left of the grey (shaded) area: Connection between some common branching vertices and a vertex x in X^i which corresponds to set $S = \{(\alpha, a'), (\beta, b'), (\gamma, c')\}$, for some $a', b', c' \in [n]$. The figure also shows vertices x_{p^α}, x_{q^α}, and x_{r^α}. On the right of the grey (shaded) area: Connection between some common vertices and element-encoding gadget encoding (α, a) for some $a \in [n]$.

The reduction repeats the constructions for elements (β, b) and (γ, c) in $\mathcal{U}$. In the first case, the vertices in the gadgets are connected to $\{p^\beta, q^\beta, r^\beta\}$ whereas in the latter case they are connected to $\{p^\gamma, q^\gamma, r^\gamma\}$.

Finally, for any vertices of the type w_a^α, w_b^β, and w_c^γ added in the above steps, the reduction connects each of them to g_4 using simple paths of length M^2 for each connection. See Figure 1.

This completes the construction. Suppose P is the collection of all the pendant vertices in G. The reduction returns $(G, k = n + |P|)$ as an instance of Geodetic Set.

Note that each edge in Figure 1 corresponds to a simple path whose length is at least $0.99 \cdot M^2$ and at most $1.01 \cdot M^2$. Hence, the distance between any two vertices in Figure 1, where we use an edge to denote a simple path connecting two of its endpoints, is proportional to their distance in G.

Define P to be the collection of all the pendants vertices in G. To prove the correctness of the reduction, we first argue that vertices in P covers most of the vertices in G. We argue that the only vertices they do not cover are internal vertices in the red edges of Figure 1. We define these uncovered vertices as follows. Define V^{g_1} as the collection of all the internal vertices in the simple path connecting g_1 to z_1^i for any $i \in [n]$. For every $a \in [n]$, define V_a^α as the collection of internal vertices in the simple path connecting q^α to u_a^α and simple path connecting u_a^α to w_a^α. Define $V^\alpha = \bigcup_{a \in [n]} V_a^\alpha$. Similarly, define V^β and V^γ.

Recall that $I(u, v)$ denotes the set of vertices in G that are part of some shortest path between u and v. Also, we generalize this definition to set S by taking the union of $I(u, v)$ for all pairs of vertices u, v in S.

▶ **Lemma 2.** *Let P be the collection of pendant vertices in G. Then, $V(G) \setminus (V^{g_1} \cup V^\alpha \cup V^\beta \cup V^\gamma) = I(P)$.*

Proof. We first prove that $V(G) \setminus (V^{g_1} \cup V^\alpha \cup V^\beta \cup V^\gamma) \subseteq I(P)$. By the construction, any vertex u in G is adjacent with at most one pendant vertex. We use $\mathsf{pndt}(u)$ to denote the pendant vertex adjacent with u, if it exists. We first consider the partition of vertices in G based on the simple paths they are part of.

Consider the simple paths ending at g_1. From the construction, $I(\mathsf{pndt}(g_1), \mathsf{pndt}(g_2))$ contains all the vertices in the simple path connecting g_1 and g_2. Similarly, $I(\mathsf{pndt}(z_2^i), \mathsf{pndt}(g_1))$ covers all the vertices in the simple path connecting z_2^i and g_1 for every $i \in [n]$. Note that the statement of the lemma excludes the vertices in simple paths connecting g_1 and z_1^i.

Consider the simple paths ending at g_2. From the construction, $I(\mathsf{pndt}(z_2^i), \mathsf{pndt}(g_2))$ covers all the vertices in the simple path connecting z_1^i to g_2 for every $i \in [n]$. Consider vertices x in X^i and y in Y^i which are the part of i^{th} set-encoding gadget $\mathcal{S}^i$ for some i in $[n]$. $I(\mathsf{pndt}(y), \mathsf{pndt}(g_2))$ covers all the vertices in the simple path connecting y to g_2. Also, $I(\mathsf{pndt}(g_2), \mathsf{pndt}(g_3))$ covers all the vertices in the path connecting g_2 with x_{p^α}. A similar statement is true for the other branching vertices adjacent to x.

Consider the simple paths ending at g_3. From the construction, $I(\mathsf{pndt}(g_3), \mathsf{pndt}(g_4))$ contains all the vertices in the simple path connecting g_3 and g_4. As argued in the previous paragraph, $I(\mathsf{pndt}(g_2), \mathsf{pndt}(g_3))$ covers all the vertices in the path connecting g_3 with all the branching vertices like x_{p^α} that are adjacent with x.

Consider the simple paths ending at g_4. For all such paths, there other endpoints are adjacent to some pendant vertex. By the construction, the shortest path between the pendant vertices adjacent to the endpoints covers all the vertices in the simple path.

Consider the simple paths ending at p^α, p^β or p^γ. All the simple paths ending at p^α can be partitioned into two parts: ones that have the other endpoint in a set-encoding gadget (i.e. x_{p^α} for some x in X^i for some $i \in [n]$) or ones that have the other endpoint in an element-encoding gadget (i.e. u_a^α for some $a \in [n]$). In the second case, both the endpoints of the simple path are adjacent to pendant vertices. Hence, by the construction, the shortest path between the pendant vertices adjacent to the endpoints cover all the vertices in the simple path. In the first case, suppose the other endpoint of the simple path is x_{p^α} for some x in X^i. Consider vertex x and the unique vertex y in Y^i which is at the distant M^2 from x. By the construction, there is a shortest path from p^α to y that contains x. Hence, $I(\mathsf{pndt}(p^\alpha), \mathsf{pndt}(y))$ covers all the vertices in the simple path connecting p^α to x. Since x is an arbitrary point in X^i, the statement is true for all the vertices in the first type of simple path. These vertices are also covered by $I(\mathsf{pndt}(p^\alpha), \mathsf{pndt}(g_3))$. A similar set of arguments implies that all the vertices in the simple path ending at p^β and p^γ are covered by vertices in P.

Consider the simple paths ending at q^α, q^β or q^γ. Once again, all the simple paths ending at q^α can be partitioned into two parts: ones that have the other endpoint in a set-encoding gadget (i.e. x_{q^α} for some x in X^i for some $i \in [n]$) or ones that have the other endpoint in an element-encoding gadget (i.e. u_a^α or v_a^α for some $a \in [n]$). Fix an integer $a \in [n]$ and recall that q_a^α is adajcent with q^α. Using the same arguments as in the above paragraph, $I(\mathsf{pndt}(q_a^\alpha), \mathsf{pndt}(y))$ covers all the vertices in the simple path connecting q^α and x. This proves the claim for the vertices in the first type of path. $I(\mathsf{pndt}(q_a^\alpha), \mathsf{pndt}(u_a^\alpha))$ covers all the vertices in the path connecting q_a^α and u_a. Note that the vertices in the simple path connecting q_a^α and v_a are in V^α, which are excluded in the statement of the lemma. A similar set of arguments implies that all the vertices in the simple path ending at q^β and q^γ are covered by vertices in P.

Consider the simple paths ending at r^α, r^β or r^γ. A similar set of arguments regarding paths ending at p^α, p^β or p^γ proves that all the vertices in simple paths ending at r^α, r^β or r^γ are covered by vertices in P.

Consider the simple paths whose endpoints are contained in a set-encoding gadget. Consider a set-encoding gadget $\mathcal{S}^i$ for a fixed i in $[n]$. The shortest paths connecting z_2^i and g_3 covers all vertices in simple paths connecting z_2^i to z_1^i and z_1^i to x for every x in X^i. As argued before, there is a shortest path from p^α to y, for any y in Y^i, that contains the corresponding x in X^i. Hence, $I(\mathsf{pndt}(p^\alpha), \mathsf{pndt}(y))$ covers all the vertices in the simple path connecting x to y. This implies that all the vertices in $\mathcal{S}^i$, except the vertices in V^{g_1} are covered. Recall that V^{g_1} contains the internal vertices of the path connecting g_1 with z_1^i.

Consider the simple paths whose both endpoints are contained in an element-encoding gadget. Consider an element-encoding gadget encoding (α, a) for some a in $[n]$. By the construction, $I(\mathsf{pndt}(u_a^\alpha), \mathsf{pndt}(w_a^\alpha))$ covers all the vertices in the simple path connecting u_a^α and w_a^α. We remark that the vertices in the simple path connecting v_a^α and w_a^α are part of V^α and hence excluded from the statement of the lemma. Using similar arguments for elements of the form (β, b) and (γ, c) for some b, c in $[n]$.

The arguments above imply that $V(G) \setminus (V^{g_1} \cup V^\alpha \cup V^\beta \cup V^\gamma) \subseteq I(P)$. It remains to prove that no vertex in $(V^{g_1} \cup V^\alpha \cup V^\beta \cup V^\gamma)$ is covered by the vertices in P.

Consider the vertices in V^{g_1} which are in the i^{th} copy of set-encoding gadget for some i in $[n]$. It is easy to see that vertices in $P \setminus \{\mathsf{pndt}(z_2^i), \mathsf{pndt}(g_1)\}$ can not cover the vertices mentioned in the previous sentence. Any shortest path whose one endpoint is $\mathsf{pndt}(z_2^i)$ and other endpoint is in $P \setminus \{\mathsf{pndt}(g_1)\}$ contains either $\{g_1, g_2\}$ or $\{z_1^i, g_2\}$. Hence, none of such paths can cover vertices in the path connecting g_1 and z_1^i. Similarly, any shortest path whose one endpoint is $\mathsf{pndt}(g_1)$ and other endpoint is in $P \setminus \{\mathsf{pndt}(z_2^i)\}$ contains g_2 and hence can not contain z_1^i. This implies that P can not cover any vertex in V^{g_1}.

Consider the vertices in V^α. It is easy to verify that vertices in V^α are not covered by the shortest path between any two common branching vertices. Also, the shortest paths between the pendant vertices in the element-encoding gadget do not cover vertices in V^α. Consider vertices in V^α, which are part of the element-encoding gadget corresponding to element (α, a) for some a in $[n]$. Any shortest path between the vertices in P that can cover these vertices has $\mathsf{pndt}(w_a^\alpha)$ as one of its endpoint and should contain q^α. By the construction, no path connecting $\mathsf{pndt}(w_a^\alpha)$ to any of the pendant vertex adjacent with common branching vertices contains q^α. It is easy to see for p^α, r^α, g_3 and g_4. For the remaining common vertices, it is sufficient to prove that the shortest path from $\mathsf{pndt}(w_a^\alpha)$ to g_2 do not contain q^α. Note that the path from w_a^α to g_2 that contains q^α and x_{q^α} for some x in X^i, is of length close to $4M^2$. However, the shortest path from $\mathsf{pndt}(w_a^\alpha)$ to g_2 is of length $3M^2$ and contains y in some Y^i and g_4. This implies that P does not cover any vertex in V^α. Using similar arguments, the statement holds for vertices in V^β and V^γ which concludes the proof of the lemma. ◀

In the following two lemmas, we prove that the reduction is safe.

▶ **Lemma 3.** *If $(\mathcal{U}, \mathcal{S})$ is a Yes-instance of 3-Dimensional Matching then (G, k) is a Yes-instance of Geodetic Set.*

Proof. Without loss of generality, suppose $\{S_1, S_2, \ldots, S_n\}$ is a collection of n sets in collection of $\mathcal{S}$ (which contains m sets) that partitions all the vertices in $\mathcal{U}$. For every i in $[n]$, consider a vertex in X^i in the set-encoding gadget $\mathcal{S}^i$ corresponding to set S_i. Define set $Q = \{x_1^1, x_2^2, \ldots, x_n^n\}$, where x_i^i is in X^i. Recall that P is the collection of all the pendant vertices in G. It is easy to see that $|P \cup Q| = k$. Due to Lemma 2, to prove that $P \cup Q$ is a geodetic set of G, it is sufficient to prove that Q covers vertices in $V^{g_1} \cup V^\alpha \cup V^\beta \cup V^\gamma$. See the paragraph above Lemma 2 for the definition of these sets. Also, recall that for any vertex u, $\mathsf{pndt}(u)$ denotes the unique pendant vertex adjacent to it, if one exists.

For a fixed i in $[n]$, $I(x_i^i, \mathsf{pndt}(g_1))$ covers all the vertices in the simple path connecting z_1^i to g_1. As this is true for any i in $[n]$, Q covers vertices in V^{g_1}. Suppose $S^i = \{(\alpha, a), (\beta, b), (\gamma, c)\}$ for some a, b, c in $[n]$. Consider vertex x_i^i in X^i. We argue that the shortest paths between x_i^i and w_a^α covers the vertices in V^α that are part of the element-encoding gadget that is encoding element (α, a). Consider Figure 2 with $a' = a$. In this case, all three paths from x_i^i to u_a^α via p^α, q^α, and q^α as the same length of $2M^2$. This implies the shortest distance between x_i^i and w_a^α is $3M^2$. Moreover, the path connecting x_i^i to w_a^α that contains q^α and v_a^α is the shortest path between these two vertices. Hence, the shortest path between x_i^i and $\mathsf{pndt}(w_a^\alpha)$ covers all the vertices in V^α that are part of element-encoding gadget. As $\{S_1, S_2, \ldots, S_n\}$ partitions $\mathcal{U}$, (exactly) one of these n set contains element (α, a) for every a in $[n]$, all the vertices in V^α are covered by a vertex in Q and a vertex in P. Using the similar arguments, all the vertices in V^β and V^γ are covered by vertices in $P \cup Q$. This concludes the proof of the lemma. ◀

▶ **Lemma 4.** *If (G, k) is a Yes-instance of Geodetic Set then $(\mathcal{U}, \mathcal{S})$ is a Yes-instance of 3-Dimensional Matching.*

Proof. Recall that P is the collection of all the pendant vertices in G, and P is a subset of any geodetic set of G. Suppose $P \cup Q$ is the geodetic set of G of size of k, and hence $|Q| \leq n$. By Lemma 2, vertices in Q covers all the vertices in $V^{g_1} \cup V^\alpha \cup V^\beta \cup V^\gamma$. See the paragraph before Lemma 2 for the definition of these sets.

We first prove that Q contains at least one vertex in the i^{th} copy of set-encoding gadget $\mathcal{S}^i$ for every i in $[n]$. Assume, for the sake of contradiction, that this is not the case for an index i. Recall that V^{g_1} is the union of internal vertices connecting paths g_1 and z_1^i for i in $[n]$. Define $V_i^{g_1}$ as the set of vertices in V^{g_1} which are in $\mathcal{S}^i$. Consider a vertex, say h, in $V_i^{g_1}$. As $P \cup Q$ is a geodetic set of G, there are two vertices in it, say s_1 and s_2, that covers the vertex h. By Lemma 2, s_1 and s_2 are *not* in P. By the above assumption, s_1 and s_2 are *not* in $\mathcal{S}^i$. Consider the shortest path from s_1 to s_2 that contains the vertex h. Suppose that the paths enter and exit $\mathcal{S}^i$ via common branching vertices c_1 and c_2. Consider the case when $\{c_1, c_2\} \cap \{q^\alpha, q^\beta, q^\gamma\} = \emptyset$. In this case, the shortest path between $\mathsf{pndt}(c_1)$ and $\mathsf{pndt}(c_2)$ also contains the vertex h. However, this contradicts Lemma 2, which states vertices in P can not cover any vertex in V^{g_1}. Consider the other case when the shortest path enters $\mathcal{S}^i$ via, say, q^α. In this case, the above arguments can be extended to the shortest path from $\mathsf{pndt}(q_a^\alpha)$ for some a in $[n]$. This also leads to contradiction to Lemma 2. Hence, our assumption is wrong and Q contains at least one vertex in $\mathcal{S}^i$.

As the cardinality of Q is at most n, this implies that Q contains exactly one vertex in $\mathcal{S}^i$ for every i in $[n]$. Let q be the unique vertex in Q which is in $\mathcal{S}^i$. We determine the position of Q in $\mathcal{S}^i$ using the fact that the shortest paths from q to some other vertices in $(P \cup Q) \setminus \{q\}$ covers all the uncovered vertices in the set-encoding gadget $\mathcal{S}^i$. We now restrict the possible cases for this other vertex in $P \cup Q$. Let h be the other vertex in $(P \cup Q) \setminus \{q\}$, so the shortest path from q to h covers some vertices in $V_i^{g_1}$. By the arguments above, h is not in $\mathcal{S}^i$. As argued in the previous paragraph, the vertices covered by the shortest path from q to h are also covered by the shortest path from q to $\mathsf{pndt}(c)$ where c is a common branching vertex or the vertex with pendant neighbor which is adjacent to either q^α, q^β, or q^γ. Hence, while considering the cases of shortest path from q to h, it is sufficient to check the cases when h is one of the vertex mentioned in the previous sentence. Note that as no vertex in Q is present in any element-encoding gadget, vertices in Q, which are in set-encoding gadgets, are also used to cover the uncovered vertices in the element-encoding gadget. We use this property to narrow the location of q in $\mathcal{S}^i$.

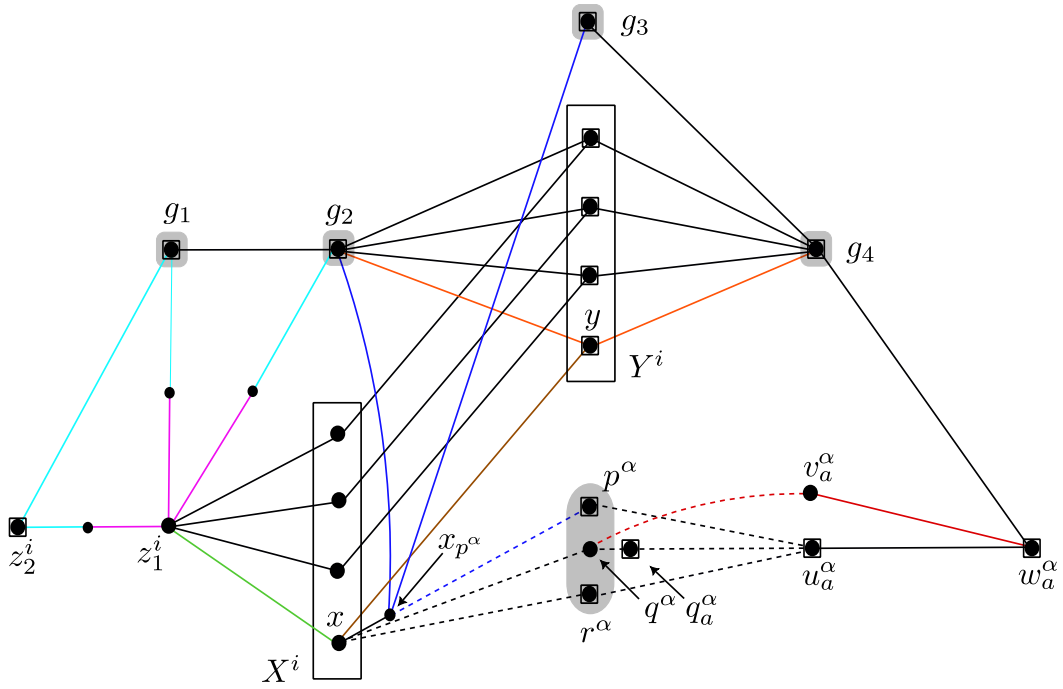


Figure 3 Types of partition of vertices in $\mathcal{S}^i$. Different colors correspond to different types of parts in the set-encoding gadget. For clarity, the figure does not show all the vertices in $\mathcal{S}^i$.

We partition $\mathcal{S}^i$ into the following parts and prove that q can be located in particular types of parts. By the construction, deleting all the common branching vertices results in a collection of trees. Consider $\mathcal{S}^i$ and root it at z_1^i.

- For a vertex x in X^i, consider the vertex x_{p^α} adjacent with it and on the path connecting p^α. Define x_{p^α}*-part* as the subtree rooted at x_{p^α}. Similarly, define x_{p^β}*-part* and x_{p^γ}*-part*. In Figure 3, x_{p^α}-part is highlighted in blue.
- For every y in Y^i, define y-part as subtree rooted at y, which is highlighted using orange color in Figure 3.
- For every x in X^i and the corresponding y in Y^i, i.e. the unique vertex in Y^i which is at distance M^2 from x, define (x, y)*-part* as the collection of internal vertices of the path from x to y. See the brown edge in Figure 3.
- For every x in X^i, define $(z_1^i, x]$*-part* as the vertex x and the internal vertices of the path from x to z_1^i. See the green part in Figure 3. Note that the part contains x but not z_1^i.
- Let h, h_1 and h_2 be the middle points of the paths from z_1^i to z_2^i, from z_1^i to g_1, and from z_2^i to g_2, respectively. Define z_1^i*-part* as the tree rooted at z_1^i with h, h_1 and h_2 as its leaves. This is highlighted by the purple color in Figure 3.
- All the remaining vertices are considered as *left-over part*, highlighted using cyan color in Figure 3. Note that this is the only disconnected part in the partition.

Using the fact that q is used to cover the vertices in $V_i^{g_1}$, we first argue it is not in the part of the first-type or of the second type mentioned above, nor can it be in the leftover part. Assume q is in x_{p^α}-part for some x in X^i. In this case, the shortest path from q to g_1 is via g_2 and hence does not contain vertices in $V_i^{g_1}$. Similarly, the shortest path from q to z_2^i is via g_1 or via z_1^i and can not contain vertices in $V_i^{g_1}$. It is easy to see that in this case, there are no paths from q to any other vertex in P can cover vertices in $V_i^{g_1}$. Hence, q cannot be

in x_{p^α}-part. Now, assume q is in y-part for some y in Y^i. The shortest path from q to g_1 is via g_2 and to z_2^i is via g_1 or z_1^i; hence, no such short paths can cover vertices in $V_i^{g_1}$. Finally, it is easy to see that there is no vertex in the leftover part can cover any vertex in $V_i^{g_1}$.

We now turn our attention to the remaining parts, viz (x,y)-part, $(z_1^i,x]$-part, for some x in X^i, or z_1^i-part. Note that the shortest path from any vertex in $(z_1^i,x]$-part or z_1^i-part to g_1 covers all the vertices in $V_i^{g_1}$ whereas the same is true for half the vertices in (x,y)-part which are closer to x than they are to y. In the next few paragraphs, we argue that the only vertices in the $(z_1^i,x]$-part can be used to cover the uncovered vertices in the element encoding gadget. Recall that for every $a \in [n]$, define V_a^α as the collection of internal vertices in the simple path connecting q^α to u_a^α and the simple path connecting u_a^α to w_a^α. We also defined $V^\alpha = \bigcup_{a\in[n]} V_a^\alpha$ and defined V^β and V^γ in a similar way. Note that V^α, V^β, and V^γ are the collection of uncovered vertices in the element-encoding gadget.

Suppose vertex x in X^i corresponds to set $S = \{(\alpha,a'),(\beta,b'),(\gamma,c')\}$ and h is a vertex in $(z_1^i,x]$-part. We prove that $I(h,w_a^\alpha)$ covers V_a^α if and only if $a' = a$. For a vertex h in $(z_1^i,x]$-part, suppose $\mathrm{dist}(h,x) = d_h$. Consider the four paths from h to w_a^α that are of length at most $d_h + 3M^2$ and contains exactly one common branching point amongst $\{g_4, p^\alpha, q^\alpha, r^\alpha\}$. Consider the path from h to w_a^α that contains x, the corresponding y, and g_4. By the construction, the length of this path is $d_h + 3M^2$. For the remaining three paths, consider Figure 2. The lengths of these three paths is $d_h + 3M^2 + 2(a'-a)M$, $d_h + 3M^2 + (a-a')M$, and $d_h + 3M^2 + 2(a-a')M$ when it contains p^α, q^α, and r^α, respectively. Note that $I(h,w_a^\alpha)$ covers V_a^α if and only the shortest path from h to w_a^α contains q^α. From the distances above, it is easy to see that this condition is true only when $a = a'$. Following the identical arguments, we can prove that $I(h,w_b^\beta)$ covers V_b^β if and only if $b' = b$, and $I(h,w_c^\gamma)$ covers V_a^γ if and only if $c' = c$.

We next prove that no vertex from (x,y)-part or z_1^i-part can cover any uncovered vertex in the element-encoding gadget. Consider a vertex h in (x,y)-part. As argued in the previous paragraph, $I(h,w_a^\alpha)$ can cover the vertices in V^α if and only if the shortest path from h to w_a^α contains q^α. However, the shortest path from h to w_a^α contains g_4 and hence cannot cover the vertices in V^α. Now consider the vertices in z_1^i-part. For any vertex h in z_1^i-part which is an internal vertex of path from z_1^i to g_2, the shortest path h to w_a^α contains g_4 and hence can not cover vertices in V^α. Note that the shortest path from z_1^i to w_a^α also contains g_4 in it. This is implied by the fact that the distance between g_2 and every vertex in Y^i is $M^2 - 1$ (instead of M^2 like most of the other paths). Hence, $I(z_1^i,w_a^\alpha)$ does not cover any vertex in V^α. For any vertex in the remaining z_1^i-part, the shortest path from it to w_a^α is via z^i, which contains g_4. These arguments imply that q can not be in (x,y)-part or z_1^i- part for any x in X^i.

We are in a position to conclude the proof of the lemma. Recall that $P \cup Q$ is a geodetic set of G. Considering the position of vertices in Q, we construct a collection of n subsets $\mathcal{S}'$ of $\mathcal{S}$ as follows: For every i in $[n]$, let x be the unique vertex in X^i such that the vertex in Q is in $(z_1^i,x]$-part of $\mathcal{S}^i$. If x corresponds to set S, then include S in $\mathcal{S}'$. It is easy to see that the cardinality of $\mathcal{S}'$ is at most n. We argue that $\mathcal{S}'$ covers all the vertices in $\mathcal{U}$. Consider an arbitrary element, say (α,a), in $\mathcal{U}$. The vertices in V^α are part of the element-encoding gadget that encodes (α,a). By above arguments, for some i in $[n]$, there is x in X^i such that $(z_1^i,x_1]$-part of $\mathcal{S}^i$ contains vertex q in Q with the property that $I(q,w_a^\alpha)$ covers vertices in V^α in the element-encoding gadget. Suppose x corresponds to the vertex $S = \{(\alpha,a'),(\beta,b'),(\gamma,c')\}$ for some $a',b',c' \in [n]$. From the arguments above, $I(q,w_a^\alpha)$ covers vertices in V^α if and only if $a' = a$. This implies that (α,a) is covered by some set in $\mathcal{S}'$. Since this is an arbitrary element in $\mathcal{U}$, one can conclude that $\mathcal{S}'$ covers all the vertices in $\mathcal{U}$. This implies that $(\mathcal{U},\mathcal{S})$ is a Yes-instance of 3-Dimensional Matching. ◀

▶ **Lemma 5.** *Pathwidth and feedback vertex set the number of G are at most* 17 *and* 13*, respectively.*

Proof. We present a mixed search strategy to clean G using 17 searchers to bound the pathwidth of G. Place 13 searchers on common branching vertices. These searchers will never move from their places. (This is equivalent to deleting common branching vertices and presenting a mixed search strategy using four searchers for the resulting graph.) We search the graph in $3n + n$ rounds, one round for each element-encoding gadget and one round for each set of set-encoding gadget, respectively.

Consider an element-encoding gadget encoding an element, say (α, a) for some a in $[n]$. We can place three searchers on u_a^α, v_a^α, and w_a^α, and the fourth searcher can move along the simple path connecting these vertices and with common branching vertices to clean all the edges in the connecting paths.

By the construction, deleting all the common branching vertices results in a collection of trees. Consider the set-encoding gadget $\mathcal{S}^i$ for some i in $[n]$ and root it at z_1^i. In each round, we fix a searcher on z_1^i. We need one searcher to clean all the edges between the simple path connecting z_1^i to g_1 via z_2^i, path of length M^2 connecting z_1^i to g_1, and finally path connecting z_1^i to g_2. Once these paths are cleaned, the searcher is free and can be used to clean the remaining part of the set-encoding gadget. We divide the remaining part of this round into m sub-rounds corresponding to each vertex in X^i. For a sub-round, fix two searchers on a vertex x in X^i and another searcher on the corresponding vertex y in Y^i which is at distance M^2 from x. The remaining searcher can clean all the edges in the simple paths connecting x to y, y to common vertices, and x to z_1^i. Next, the searcher placed on y can be relocated to each of the branching vertices adjacent to x, i.e. vertices of the form x_{p^α}. Now, the other searcher can be used to clean the subtree rooted at the branching vertex. This completes one sub-round. Completing such m sub-rounds clears all the paths in $\mathcal{S}^i$.

As we need 17 searchers to clean the graph, result of Takahashi et al. [38] implies that $\mathtt{pw}(G) \leq 17$.

As deleting all the common branching vertices results in a collection of trees, feedback vertex set number of the graph is 13. ◀

4 Conclusion

In this article, we proved that Geodetic Set is NP-complete even on graphs with constant pathwidth and feedback vertex set number answering the open question by Kellerhals and Koana [34]. In the same paper, the authors provided an FPT algorithm (with impractical running time) when parameterized by treedepth (denoted by $\mathtt{td}$) of the input graph. Recent work by Foucaud et al. [23] showed that Geodetic Set admits an algorithm running in time $2^{\mathtt{diam}^{\mathcal{O}(\mathtt{tw})}} \cdot poly(|V(G)|)$, where $\mathtt{diam}$ denotes the diameter of the graph. This result, along with the fact that $\mathtt{diam}$ and $\mathtt{tw}$ are upper bounded by $2^{\mathtt{td}}$ and $\mathtt{td} + 1$, respectively, imply that Geodetic Set admits an algorithm running in time $2^{2^{\mathcal{O}(\mathtt{td}^2)}} \cdot poly(|V(G)|)$. It would be interesting to obtain an FPT algorithm parameterized by treedepth with improved running time. We remark that Foucaud et al. [23] proved that the problem does not admit an algorithm running in time $2^{2^{o(\mathtt{td})}} \cdot poly(|V(G)|)$, unless the ETH fails.

As highlighted in the introduction of this article and also in [8, 34], Metric Dimension and Geodetic Set share many hardness properties. We mention one notable exception: Metric Dimension is FPT when parameterized by the solution size plus the treelength [3], however, Geodetic Set is W[2]-hard when parameterized by the solution size even when treelength is a constant. See [17, Theorem 2]. Regarding the parameterized complexity of

Metric Dimension when parameterized by pathwidth and feedback vertex set number of the graph, Galby et al. [25] showed that the problem is W[1]-hard. Can we improve this result to obtain a result for Metric Dimension as we did for Geodetic Set in this article?

References

1 J. Ahn, L. Jaffke, O. Kwon, and P. T. Lima. Well-partitioned chordal graphs. *Discrete Mathematics*, 345(10):112985, 2022. doi:10.1016/J.DISC.2022.112985.

2 M. Atici. Computational complexity of geodetic set. *International journal of computer mathematics*, 79(5):587–591, 2002. doi:10.1080/00207160210954.

3 R. Belmonte, F. V. Fomin, P. A. Golovach, and M. S. Ramanujan. Metric dimension of bounded tree-length graphs. *SIAM J. Discrete Math.*, 31(2):1217–1243, 2017. doi:10.1137/16M1057383.

4 B. Bergougnoux, O. Defrain, and F. Mc Inerney. Enumerating minimal solution sets for metric graph problems. In *Graph-Theoretic Concepts in Computer Science - 50th International Workshop, WG*, 2024. doi:10.1007/978-3-031-75409-8_4.

5 E. Bonnet and N. Purohit. Metric dimension parameterized by treewidth. *Algorithmica*, 83:2606–2633, 2021. doi:10.1007/S00453-021-00808-9.

6 F. Buckley and F. Harary. Geodetic games for graphs. *Quaestiones Mathematicae*, 8(4):321–334, 1985. doi:10.1080/16073606.1985.9631921.

7 L. R. Bueno, L. D. Penso, F. Protti, V. R. Ramos, D. Rautenbach, and U. S. Souza. On the hardness of finding the geodetic number of a subcubic graph. *Inf. Process. Lett.*, 135:22–27, 2018. doi:10.1016/J.IPL.2018.02.012.

8 D. Chakraborty, S. Das, F. Foucaud, H. Gahlawat, D. Lajou, and B. Roy. Algorithms and complexity for geodetic sets on planar and chordal graphs. In *31st International Symposium on Algorithms and Computation (ISAAC)*, 2020. doi:10.4230/LIPIcs.ISAAC.2020.7.

9 D. Chakraborty, F. Foucaud, H. Gahlawat, S. K. Ghosh, and B. Roy. Hardness and approximation for the geodetic set problem in some graph classes. In *Proc. of the 6th International Conference on Algorithms and Discrete Applied Mathematics (CALDAM)*, 2020.

10 D. Chakraborty, F. Foucaud, and A. Hakanen. Distance-based covering problems for graphs of given cyclomatic number. In *Fundamentals of Computation Theory - 24th International Symposium, FCT*, 2023.

11 D. Chakraborty, F. Foucaud, D. Majumdar, and P. Tale. Tight (double) exponential bounds for identification problems: Locating-dominating set and test cover. In *35th International Symposium on Algorithms and Computation, ISAAC*, 2024.

12 D. Chakraborty, H. Gahlawat, and B. Roy. Algorithms and complexity for geodetic sets on partial grids. *Theoretical Computer Science*, 979:114217, 2023. doi:10.1016/J.TCS.2023.114217.

13 G. Chartrand, F. Harary, and P. Zhang. On the geodetic number of a graph. *Networks*, 39(1):1–6, 2002. doi:10.1002/NET.10007.

14 M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.

15 T. Davot, L. Isenmann, and J. Thiebaut. On the approximation hardness of geodetic set and its variants. In *Proc. of the 27th International Computing and Combinatorics Conference, COCOON*, 2021.

16 R. Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

17 M. Dourado, F. Protti, D. Rautenbach, and J. Szwarcfiter. Some remarks on the geodetic number of a graph. *Discret. Math.*, 310(4):832–837, 2010. doi:10.1016/J.DISC.2009.09.018.

18 M. C. Dourado, F. Protti, D. Rautenbach, and J. L. Szwarcfiter. On the complexity of the geodetic and convexity numbers of a graph. In *Proc. of the International Conference on Discrete Mathematics (ICDM)*, volume 7 of *RMS Lecture Notes Series*, pages 101–108. Ramanujan Mathematical Society, 2008.

19 M. C. Dourado, F. Protti, D. Rautenbach, and J. L. Szwarcfiter. Some remarks on the geodetic number of a graph. *Discrete Mathematics*, 310(4):832–837, 2010. doi:10.1016/J.DISC.2009.09.018.

20 A. L. Douthat and M. C. Kong. Computing geodetic bases of chordal and split graph. *Journal of Combinatorial Mathematics and Combinatorial Computing*, pages 67–77, 1996.

21 T. Ekim, A. Erey, P. Heggernes, P. van't Hof, and D. Meister. Computing minimum geodetic sets of proper interval graphs. In *Proc. of the 10th Latin American Symposium on Theoretical Informatics (LATIN 2012)*, 2012.

22 M. Farber and R. E. Jamison. Convexity in graphs and hypergraphs. *SIAM Journal on Algebraic Discrete Methods*, 7(3):433–444, 1986.

23 F. Foucaud, E. Galby, L. Khazaliya, S. Li, F. Mc Inerney, R. Sharma, and P. Tale. Problems in NP can admit double-exponential lower bounds when parameterized by treewidth or vertex cover. In *51st International Colloquium on Automata, Languages, and Programming, ICALP*, 2024. doi:10.4230/LIPIcs.ICALP.2024.66.

24 F. Foucaud, E. Galby, L. Khazaliya, S. Li, F. Mc Inerney, R. Sharma, and P. Tale. Metric dimension and geodetic set parameterized by vertex cover. In *42nd International Symposium on Theoretical Aspects of Computer Science, STACS*, 2025. doi:10.4230/LIPIcs.STACS.2025.33.

25 E. Galby, L. Khazaliya, F. Mc Inerney, R. Sharma, and P. Tale. Metric dimension parameterized by feedback vertex set and other structural parameters. *SIAM J. Discrete Math.*, 37(4):2241–2264, 2023. doi:10.1137/22M1510911.

26 M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

27 F. Harary, E. Loukakis, and C. Tsouros. The geodetic number of a graph. *Mathematical and Computer Modelling*, 17(11):89–95, 1993.

28 S. Hartung and A. Nichterlein. On the parameterized and approximation hardness of metric dimension. In *Proc. of the 28th Conference on Computational Complexity, CCC 2013*, pages 266–276. IEEE Computer Society, 2013. doi:10.1109/CCC.2013.36.

29 L. Jaffke, O j. Kwon, T. J. F. Strømme, and J. A. Telle. Mim-width III. Graph powers and generalized distance domination problems. *Theoretical Computer Science*, 796:216–236, 2019. doi:10.1016/J.TCS.2019.09.012.

30 M. Kanté, T. Marcilon, and R. M. Sampaio. On the parameterized complexity of the geodesic hull number. *Theor. Comput. Sci.*, 791:10–27, 2019. doi:10.1016/J.TCS.2019.05.005.

31 M. M. Kanté and L. Nourine. Polynomial time algorithms for computing a minimum hull set in distance-hereditary and chordal graphs. *SIAM J. Discrete Math.*, 30(1):311–326, 2016. doi:10.1137/15M1013389.

32 I. Katsikarelis, M. Lampis, and V. Th. Paschos. Structural parameters, tight bounds, and approximation for (k, r)-center. *Discret. Appl. Math.*, 264:90–117, 2019. doi:10.1016/J.DAM.2018.11.002.

33 I. Katsikarelis, M. Lampis, and V. Th. Paschos. Structurally parameterized d-scattered set. *Discrete Applied Mathematics*, 308:168–186, 2022. doi:10.1016/J.DAM.2020.03.052.

34 L. Kellerhals and T. Koana. Parameterized complexity of geodetic set. *Journal of Graph Algorithms and Applications*, 26(4):401–419, 2022. doi:10.7155/JGAA.00601.

35 S. Li and M. Pilipczuk. Hardness of metric dimension in graphs of constant treewidth. *Algorithmica*, 84(11):3110–3155, 2022. doi:10.1007/S00453-022-01005-Y.

36 M. Mezzini. Polynomial time algorithm for computing a minimum geodetic set in outerplanar graphs. *Theoretical Computer Science*, 745:63–74, 2018. doi:10.1016/J.TCS.2018.05.032.

37 I. M. Pelayo. *Geodesic Convexity in Graphs*. Springer, 2013.

38 A. Takahashi, S. Ueno, and Y. Kajitani. Mixed searching and proper-path-width. *Theor. Comput. Sci.*, 137(2):253–268, 1995. doi:10.1016/0304-3975(94)00160-K.

Hitting Geodesic Intervals in Structurally Restricted Graphs

Tatsuya Gima
Hokkaido University, Sapporo, Japan

Yasuaki Kobayashi
Hokkaido University, Sapporo, Japan

Yuto Okada
Nagoya University, Japan

Yota Otachi
Nagoya University, Japan

Hayato Takaike
Nagoya University, Japan

Abstract

Given a graph $G = (V, E)$, a set T of vertex pairs, and an integer k, Hitting Geodesic Intervals asks whether there is a set $S \subseteq V$ of size at most k such that for each terminal pair $\{u, v\} \in T$, the set S intersects at least one shortest u–v path. Aravind and Saxena [WALCOM 2024] introduced this problem and showed several parameterized complexity results. In this paper, we extend the known results in both negative and positive directions and present sharp complexity contrasts with respect to structural graph parameters.

We first show that the problem is NP-complete even on graphs with highly restricted shortest-path structures. More precisely, we show the NP-completeness on graphs obtained by adding a single vertex to a disjoint union of 5-vertex paths. By modifying the proof of this result, we also show the NP-completeness on graphs obtained from a path by adding one vertex and on graphs obtained from a disjoint union of triangles by adding one universal vertex. Furthermore, we show the NP-completeness on graphs of bandwidth 4 and maximum degree 5 by replacing the universal vertex in the last case with a long path. Under standard complexity assumptions, these negative results rule out fixed-parameter algorithms for most of the structural parameters studied in the literature (if the solution size k is not part of the parameter).

We next present fixed-parameter algorithms parameterized by k plus modular-width and by k plus vertex integrity. The algorithm for the latter case does indeed solve a more general setting that includes the parameterization by the minimum vertex multiway-cut size of the terminal vertices. We show that this is tight in the sense that the problem parameterized by the minimum vertex multicut size of the terminal pairs is W[2]-complete. We then modify the proof of this intractability result and show that the problem is W[2]-complete parameterized by k even in the setting where $T = \binom{Q}{2}$ for some $Q \subseteq V$.

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

Keywords and phrases Terminal monitoring set, Structural graph parameter, Geodesic interval

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.29

Related Version *Full Version*: https://arxiv.org/abs/2509.01413

Funding *Tatsuya Gima*: JSPS KAKENHI Grant Numbers JP24K23847, JP25K03077
Yasuaki Kobayashi: JSPS KAKENHI Grant Numbers JP23K28034, JP24H00686, JP24H00697
Yuto Okada: Supported by JST SPRING, Grant Number JPMJSP2125
Yota Otachi: JSPS KAKENHI Grant Numbers JP22H00513, JP24H00697, JP25K03076, JP25K03077

© Tatsuya Gima, Yasuaki Kobayashi, Yuto Okada, Yota Otachi, and Hayato Takaike;
licensed under Creative Commons License CC-BY 4.0

1 Introduction

For vertices u and v in a graph G, the *geodesic interval* $I_G[u, v]$ is the set of vertices that belong to at least one shortest u–v path in G. In this paper, we study the problem Hitting Geodesic Intervals that asks to find a minimum hitting set for a given family of geodesic intervals, which is formalized as follows.

Problem: Hitting Geodesic Intervals (HGI)
Input: A graph $G = (V, E)$, a set T of vertex pairs, and an integer k.
Question: Is there $S \subseteq V$ with $|S| \leq k$ such that $S \cap I_G[u, v] \neq \emptyset$ for each $\{u, v\} \in T$?

We also study the same problem on edge-weighted graphs, which we call Weighted Hitting Geodesic Intervals (WHGI). We call a vertex belonging to $V(T) = \bigcup_{\{u,v\}\in T}\{u, v\}$ a *terminal vertex*. We say that a vertex w *interrupts* a pair $\{u, v\}$ if $w \in I_G[u, v]$. Also, we say that a vertex set S *interrupts* $\{u, v\}$ if one of the members of S interrupts $\{u, v\}$.

The setting of HGI may have many practical applications. For example, it can be seen as a facility location problem in a network G with a set T of departure-destination point pairs of transportation demands. The goal here is to locate as small number of facilities (e.g., gas stations) as possible in such a way that, for each demand $\{u, v\} \in T$, one can take a shortest u–v path in G that visits at least one facility.

HGI has been introduced recently by Aravind and Saxena [2], who motivated the setting as a communication network monitoring problem (and called it Terminal Monitoring Set). They initiated the study of the parameterized complexity of HGI and showed several positive and negative results (see Section 1.1). Our contribution in this paper is to considerably tighten the known tractability/intractability border of HGI with respect to structural graph parameters, as we describe in Section 1.2.

1.1 Known results

Observations. To illustrate connections to other problems, we first observe that some known facts imply (in)tractability of some cases of HGI. (Some of them will be useful later.)

1. WHGI parameterized by $|T|$ is fixed-parameter tractable.
2. WHGI on trees is polynomial-time solvable.
3. If Vertex Cover is NP-complete on a graph class $\mathcal{C}$, then so is HGI.
4. HGI on complete graphs is NP-complete.
5. HGI on $r \times r$ grids is NP-complete.

The first one is an easy observation that this is a special case of Hitting Set parameterized by the number of subsets to be hit, which is fixed-parameter tractable (see [8, Theorem 6.1]). The second follows from the fact that the following more general problem is polynomial-time solvable (see, e.g., the discussion by Jansen [19]): given a set of subtrees of a tree, find a minimum vertex set that intersects all subtrees in the set.

The third and fourth observations come from the equivalence of an instance $\langle (V, E), k\rangle$ of Vertex Cover and the instances $\langle (V, E), \binom{V}{2}, k\rangle$ and $\langle (V, \binom{V}{2}), E, k\rangle$ of HGI, where $\binom{V}{2} = \{\{u, v\} \mid u, v \in V, u \neq v\}$. The equivalence holds since the only way to interrupts an adjacent pair is to pick one of them.

The fifth fact can be shown by observing an interesting connection to a geometric problem. Piercing Rectangles asks, given axis-parallel rectangles $R_1, \ldots, R_r$ in the plane and an integer k, whether there is a set of k points that intersects all rectangles R_i (see [1]). It

is known that even if all coordinates are integers, each rectangle is a 2×2 square, and all rectangles are placed in a region of $cr \times cr$ for some constant c, this problem is NP-complete [16].[1] Such an instance of Piercing Rectangles can be transformed to an equivalent instance $\langle G, T, k \rangle$ of HGI by setting G to the $cr \times cr$ grid and adding the pair of two opposite corners of each rectangle R_i to T. Observe that a geodesic interval $I_G[u, v]$ for u and v is the set of vertices in the axis-parallel rectangle having u and v at opposite corners.

Results of Aravind and Saxena [2]. Their main results can be summarized as follows.

1. HGI is W[2]-complete parameterized by the solution size k.
2. HGI is fixed-parameter tractable parameterized by k plus cluster vertex deletion number.
3. HGI is fixed-parameter tractable parameterized by k plus neighborhood diversity.
4. WHGI is fixed-parameter tractable parameterized by feedback edge set number.
5. WHGI is fixed-parameter tractable parameterized by vertex cover number.

Their first result complements the fixed-parameter tractability with respect to $|T|$ as $k < |T|$ in nontrivial instances. Note that this result implies the W[2]-completeness of WHGI parameterized by k even on complete graphs (consider the reduction from HGI that replaces each nonedge with a huge-weight edge). In their second and third results, having k in the parameter is necessary because of the NP-completeness of HGI on complete graphs. Their fourth result generalizes polynomial-time solvability on trees. Note that they did not explicitly mention the weighted case for feedback edge set number, but their algorithm works without any modifications for the weighted case as well.

Since vertex cover number is an upper bound on the solution size k, the parameterization solely by vertex cover number is equivalent to the one by k plus vertex cover number. Thus, most of the known positive results (except for the one parameterized by feedback edge set number) have "plus k" in their parameters.

As open questions, Aravind and Saxena [2] asked for the complexity of HGI parameterized by treewidth, pathwidth, feedback vertex set number, and distance to path forest.

1.2 Our results

NP-completeness of HGI on highly restricted graphs. Our first results demonstrate that HGI is NP-complete even if the target graph classes are highly restricted, and thus answer the open questions of Aravind and Saxena [2] in a negative way. More precisely, we show that the problem is NP-complete on the following classes:

1. graphs of vertex-deletion distance 1 to the class of disjoint unions of 5-vertex paths;
2. graphs of vertex-deletion distance 1 to the class of paths;
3. graphs of vertex-deletion distance 1 to the class of disjoint unions of triangles;
4. graphs of bandwidth 4 and maximum degree 5.

Our main technical contribution here is the first one, which already implies that HGI is paraNP-complete for most of the structural parameters studied in the literature (see Figure 1 (left)) as the graphs there have distance to path forest 1 and vertex integrity 6. The second one is an easy corollary of the first. The third one, which improves the vertex integrity bound to 4, is based on the same idea as the first. The fourth one can be obtained from the third by a minor modification.

[1] In [16], the problem is described as a point-set covering problem by 2×2 squares, which is equivalent to the one described here because in each problem one object hits or covers another object if the L_∞ distance between their centers is at most 1.

FPT with k plus structural parameters. To cope with the rather hopeless situation discussed above, we consider the setting where the solution size k is also part of the parameter. In this setting, we present the following positive results.

1. HGI is fixed-parameter tractable parameterized by k plus modular-width.
2. WHGI is fixed-parameter tractable parameterized by k plus vertex integrity.
3. WHGI is fixed-parameter tractable parameterized by the minimum vertex multiway-cut size of the set of terminal vertices $V(T)$.

The first result for k plus modular-width generalizes the one for k plus neighborhood diversity [2]. The second and third results are achieved by a single algorithm that solves WHGI when a small separator of a special kind is given as part of the input. After presenting the algorithm, we show that the two cases can be handled by this algorithm without the assumption that the separator is given as part of the input (see Section 4.2.2 for details). Observe that the second result generalizes the one parameterized solely by vertex cover number [2] since vertex cover number is an upper bound of both k and vertex integrity. The third result generalizes the one parameterized by $|T|$, which can be seen as an example of the conceptual framework of Jansen and Swennenhuis [20] for parameterizing terminal-involved problems with a parameter structurally smaller than the number of terminals. Although the third result does not explicitly have k in the parameter, we can see that its parameter is an upper bound of k.

Figure 1 (right) shows the complexity of HGI parameterized by solution size k plus a structural parameter. As we can see, it is not well understood yet, and many cases remain unsettled.

Parameterized intractability beyond k. As mentioned above, there are many unsettled cases in the "plus k" setting. Actually, we do not know any intractability result like "HGI is W-hard parameterized by k plus X" for any structural graph parameter X.

As intermediate steps toward this direction, we present two intractability results for the following cases, where k is part of the parameter, implicitly or explicitly.

1. HGI is W[2]-complete parameterized by the minimum vertex multicut size of the set of terminal pairs T.
2. HGI is W[2]-complete parameterized by k even if $T = \binom{Q}{2}$ for some $Q \subseteq V$.

The first one implicitly has k in the parameter since such a cut size is an upper bound of k. Also, it is a lower bound of the minimum vertex multiway-cut size of $V(T)$. In this sense, this intractability result tightens the gap between the W[2]-completeness parameterized solely by k [2] and our fixed-parameter tractability result parameterized by the minimum vertex multiway-cut size of $V(T)$.

The second result shows that a natural restriction of HGI is still W[2]-complete parameterized solely by k, where we want to hit pairwise geodesic intervals among a given vertex subset $Q \subseteq V$. This result is shown by a modification of the proof of the first result.

1.3 Related work

In the fields of graph theory and graph algorithms, the concept of geodesic intervals (also known as *geodetic intervals* or just as *intervals*) is well studied, especially in the context of the problem Geodetic Set [18] (see the survey on Geodetic Set [5] and the references therein). In a graph $G = (V, E)$, a set $S \subseteq V$ is a *geodetic set* if $\bigcup_{\{u,v\} \in \binom{S}{2}} I_G[u, v] = V$; that is, the union of pairwise geodesic intervals among the vertices in S covers all vertices of G.

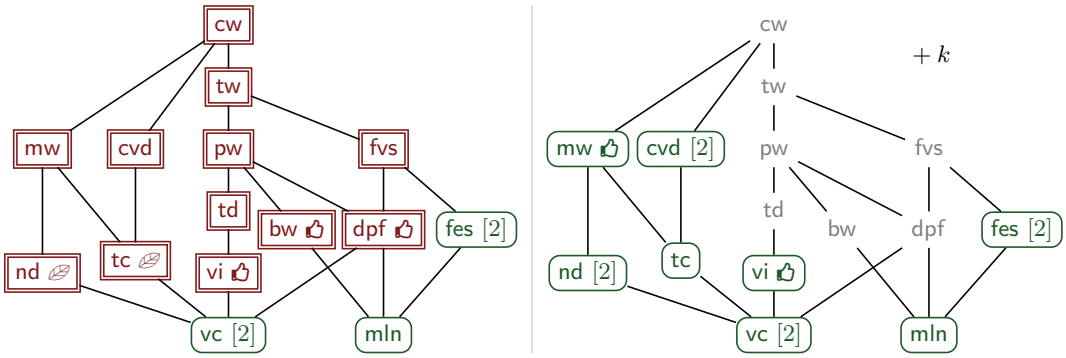


Figure 1 The figure on the left shows the complexity of Hitting Geodesic Intervals parameterized solely by each parameter, and the one on the right shows the complexity parameterized by each parameter plus solution size k. (See Section 2 for the non-abbreviated names of the parameters.) The results marked with 👍 are shown in this paper and the ones with 🍃 are easy observations. The double rectangles and the rounded rectangles indicate paraNP-complete and fixed-parameter tractable cases, respectively. Ones without frames remain unsettled. A connection between two parameters means that if the one below (say β) is bounded, then so is the one above (say α); that is, there is a function f such that $\alpha(G) \leq f(\beta(G))$ for every graph G. Thus, a positive result propagates downward, while a negative result propagates upward.

Given a graph G and an integer k, Geodetic Set asks whether G has a geodetic set of size at most k. Geodetic Set is NP-complete [3, 10] and several parameterized complexity results have been shown recently [21, 28]. Although Hitting Geodesic Intervals and Geodetic Set share some concepts in common, there seem to be no direct connections.

2 Preliminaries

Throughout the paper, we assume that the reader is familiar with the concepts in parameterized complexity and fixed-parameter tractability (see standard textbooks, e.g., [8, 11, 12, 15, 25]).

All graphs considered in this paper are finite, simple, and undirected. In Section 4.2, we consider edge-weighted graphs, where the weight of each edge represents its length. To guarantee the existence of shortest paths, we assume that the input graph does not contain a cycle of a negative total weight. It is well known that even in the edge-weighted setting, we can compute all-pairs shortest paths (or find a negative cycle) in polynomial time using standard algorithms (see e.g., [7]).

As mentioned in the introduction, WHGI is a special case of Hitting Set. Given a finite set U, a family $\mathcal{F} \subseteq 2^U$, and an integer k, Hitting Set asks whether there is $S \subseteq U$ with $|S| \leq k$ such that $S \cap F \neq \emptyset$ for each $F \in \mathcal{F}$. We can see that an instance $\langle G = (V, E), T, k \rangle$ of WHGI is equivalent to the instance $\langle V, \{I_G[u, v] \mid \{u, v\} \in T\}, k \rangle$ of Hitting Set, where the family $\{I_G[u, v] \mid \{u, v\} \in T\}$ can be computed in polynomial time. Hitting Set is known to be W[2]-complete parameterized by k, while it is fixed-parameter tractable parameterized by $|U|$ and by $|\mathcal{F}|$ (see [8]).

Although our results involve many structural graph parameters as shown in Figure 1, not all of the parameters need their formal definitions for the presentations in this paper. Thus, we only define some of them when it is necessary. For the definitions and relationships of the parameters, we refer the reader to the survey on structural graph parameters [27] (see also [26, 29]).

In Figure 1, we abbreviated the names of graph parameters as follows: cw is cliquewidth; tw is treewidth; pw is pathwidth; td is treedepth; vi is vertex integrity; vc is vertex cover number; mw is modular-width; nd is neighborhood diversity; cvd is cluster vertex deletion number; tc is twin cover number; bw is bandwidth; mln is max-leaf number; fvs is feedback vertex set number; fes is feedback edge set number; and dpf is distance to path forest.

3 NP-completeness on highly restricted graphs

Since HGI clearly belongs to NP, we only prove the NP-hardness in the following proofs.

▶ **Theorem 3.1.** *Hitting Geodesic Intervals is* NP*-complete on graphs of vertex-deletion distance* 1 *to the class of disjoint unions of* 5*-vertex paths.*

Proof. We present a polynomial-time reduction from 3-Coloring, which is NP-complete [17, GT4]. Given a graph $G = (V, E)$, 3-Coloring asks whether there is a *proper* 3*-coloring* of G, i.e., a mapping $c\colon V \to \{1, 2, 3\}$ such that $c(u) \neq c(v)$ for every $\{u, v\} \in E$.

Let $G = (V, E)$ be an instance of 3-Coloring. In the following, we construct an equivalent instance $\langle H, T, k\rangle$ of Hitting Geodesic Intervals.

Construction of $\langle H, T, k\rangle$. We set $k = 2|V|$. For each vertex $v \in V$, we construct a path P_v of five vertices v_1, v'_1, v_2, v'_3, and v_3 that appear in this order along the path. To complete the construction of H, we add one more vertex, s^*, that is adjacent to almost all other vertices except for the center vertex v_2 in each path P_v for each $v \in V$. See Figure 2. Clearly, $H - s^*$ is a disjoint union of 5-vertex paths. For each $v \in V$, we add pairs $\{v_1, v'_1\}$ and $\{v_3, v'_3\}$ into T. For each $\{u, v\} \in E$, we add pairs $\{u_1, v_1\}$, $\{u_2, v_2\}$, and $\{u_3, v_3\}$ into T.

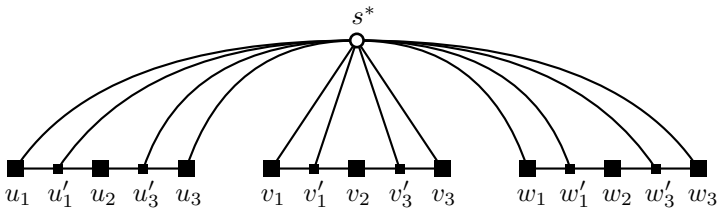


Figure 2 The construction of H in Theorem 3.1 (when G has only three vertices u, v, w). We represent terminal vertices by squares. Some terminal vertices are depicted as small squares since they appear only in "local" terminal pairs.

If G is a yes instance. Let $c\colon V \to \{1, 2, 3\}$ be a proper 3-coloring of G. For each $v \in V$, let $S_v \subseteq \{v_1, v'_1, v_2, v'_3, v_3\}$ be the following set depending on $c(v)$:

$$S_v = \begin{cases} \{v'_1, v_3\} & \text{if } c(v) = 1, \\ \{v_1, v_3\} & \text{if } c(v) = 2, \\ \{v_1, v'_3\} & \text{if } c(v) = 3. \end{cases}$$

We set $S = \bigcup_{v \in V} S_v$. Since $|S| = 2|V| = k$, it suffices to show that S interrupts all pairs in T. For each $v \in V$, it is easy to see that S_v interrupts the pairs $\{v_1, v'_1\}$ and $\{v_3, v'_3\}$.

For each edge $\{u, v\} \in E$, we show that $S_u \cup S_v$ interrupts the pairs $\{u_1, v_1\}$, $\{u_2, v_2\}$, and $\{u_3, v_3\}$. Since c is a proper 3-coloring, we have $c(u) \neq c(v)$, and thus $\{c(u), c(v)\} \cap \{1, 3\} \neq \emptyset$. By symmetry, we may assume that $c(u) \in \{1, 3\}$. Assume that $c(u) = 1$. Then, $S_u = \{u'_1, u_3\}$

interrupts $\{u_3, v_3\}$ and also $\{u_2, v_2\}$ as $u'_1 \in I_H[u_2, v_2]$. Since $c(v) \in \{2, 3\}$, S_v includes v_1, and thus it interrupts $\{u_1, v_1\}$. The other case of $c(u) = 3$ can be handled in the same way by swapping 1 and 3 in the discussion above.

If $\langle H, T, k\rangle$ is a yes instance. Let $S \subseteq V(H)$ be a set with $|S| \leq k$ that interrupts all pairs in T. Since $\{v_1, v'_1\}, \{v_3, v'_3\} \in E(H) \cap T$ for every $v \in V$ and $k = 2|V|$, we have $|S \cap \{v_1, v'_1\}| = |S \cap \{v_3, v'_3\}| = 1$ for each $v \in V$ and $S \subseteq \bigcup_{v\in V}\{v_1, v'_1, v_3, v'_3\}$. Let $S_v = S \cap \{v_1, v'_1, v_3, v'_3\}$. We define $c(v) \in \{1, 2, 3\}$ for each $v \in V$ as follows:

$$c(v) = \begin{cases} 1 & \text{if } v_1 \notin S_v, \\ 3 & \text{else if } v_3 \notin S_v, \\ 2 & \text{otherwise } (S_v = \{v_1, v_3\}). \end{cases}$$

We show that c is a proper 3-coloring of G. Suppose to the contrary that there is an edge $\{u, v\} \in E$ with $c(u) = c(v) = x$. If $x \in \{1, 3\}$, then $u_x, v_x \notin S_u \cup S_v$. Since $I_H[u_x, v_x] = \{u_x, s^*, v_x\}$ and $s^* \notin S$, S does not interrupt the pair $\{u_x, v_x\} \in T$. If $x = 2$, then $S_u \cup S_v = \{u_1, u_3, v_1, v_3\}$ does not intersect $I_H[u_2, v_2] = \{u_2, u'_1, u'_3, s^*, v'_1, v'_3, v_2\}$, and thus S ($\not\ni s^*$) does not interrupt the pair $\{u_2, v_2\} \in T$. ◀

To the graph H constructed in the proof of Theorem 3.1, we may add some vertices and edges without invalidating the reduction as long as the geodesic intervals to be hit do not change. In particular, we can pick arbitrary two vertices u and v of G and add a long path between u_3 and v_1 to H.[2] Repeating this in an appropriate way, we can make $H - s^*$ a path and obtain the following corollary.

▶ **Corollary 3.2.** *Hitting Geodesic Intervals is* NP*-complete on graphs of vertex-deletion distance* 1 *to the class of paths.*

The next theorem can be shown by replacing 5-vertex paths in the proof of Theorem 3.1 with triangles.

▶ **Theorem 3.3.** *Hitting Geodesic Intervals is* NP*-complete on graphs of vertex-deletion distance* 1 *to the disjoint unions of triangles.*

Proof. Let $G = (V, E)$ be an instance of 3-Coloring. We set $k = 2|V|$. For each $v \in V$, the graph H contains a triangle C_v with vertices v_1, v_2, and v_3. Additionally, H has vertex s^* adjacent to all other vertices. See Figure 3 (left). For each $v \in V$, we add pairs $\{v_1, v_2\}$, $\{v_2, v_3\}$, and $\{v_3, v_1\}$ into T. For each $\{u, v\} \in E$, we add pairs $\{u_1, v_1\}$, $\{u_2, v_2\}$, and $\{u_3, v_3\}$ into T.

Assume that G admits a proper 3-coloring c. For each $v \in V$, let $S_v = \{v_1, v_2, v_3\} \setminus \{v_{c(v)}\}$. For each $v \in V$, S_v interrupts the pairs $\{v_1, v_2\}$, $\{v_2, v_3\}$, and $\{v_3, v_1\}$. For each edge $\{u, v\} \in E$, since $c(u) \neq c(v)$, $S_u \cup S_v$ includes at least one of u_i and v_i for each $i \in \{1, 2, 3\}$, and thus, $S_u \cup S_v$ interrupts the pairs $\{u_1, v_1\}$, $\{u_2, v_2\}$, and $\{u_3, v_3\}$. Since $\bigcup_{v\in V} S_v$ has size $2|V| = k$, $\langle H, T, k\rangle$ is a yes instance of HGI.

Next assume that $\langle H, T, k\rangle$ is a yes instance of HGI, and that $S \subseteq V(H)$ interrupts all pairs in T and has size at most k. Since $\{v_1, v_2\}, \{v_2, v_3\}, \{v_1, v_1\} \in E(H) \cap T$ and $k = 2|V|$, we have $s^* \notin S$ and $|S \cap \{v_1, v_2, v_3\}| = 2$ for each $v \in V$. We define $c(v) \in \{1, 2, 3\}$ for each $v \in V$ to be the unique index $i \in \{1, 2, 3\}$ such that $v_i \notin S$. Suppose that there is an edge $\{u, v\} \in E$ with $c(u) = c(v) = x$. The definition of c implies that $u_x, v_x \notin S \cap \{u_1, u_2, u_3, v_1, v_2, v_3\}$. Since $s^* \notin S$ and $I_H[u_x, v_x] = \{u_x, s^*, v_x\}$, S does not interrupt the pair $\{u_x, v_x\} \in T$. ◀

[2] Actually, we can show that adding an edge between u_3 and v_1 is good enough.

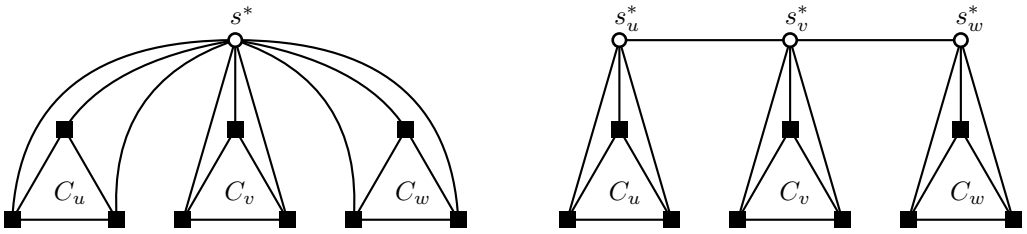


Figure 3 The construction of H and H' in Theorem 3.3 and Corollary 3.4.

To show the hardness on graphs of bandwidth 4 and maximum degree 5, we only need to slightly modify the proof of Theorem 3.3. For an n-vertex graph $G = (V, E)$, its bandwidth $\mathsf{bw}(G)$ is defined as $\mathsf{bw}(G) = \min_\sigma \max_{\{u,v\}\in E} |\sigma(u) - \sigma(v)|$, where the minimum is taken overall bijections $\sigma\colon V \to [n]$, where $[n] = \{i \in \mathbb{Z} \mid 1 \le i \le n\}$.

▶ **Corollary 3.4.** *Hitting Geodesic Intervals is* NP*-complete on graphs of bandwidth* 4 *and maximum degree* 5*.*

Proof. Let H be the graph constructed in the proof of Theorem 3.3. We first remove s^* from H. Next, for each $v \in V$, we add a vertex s^*_v adjacent to all vertices in the triangle C_v. Finally, we add $|V| - 1$ edges to make a path that goes through the new vertices $\{s^*_v \mid v \in V\}$. See Figure 3 (right). Let us call the modified graph H'. We can see that $\langle H, T, k\ (= 2|V|)\rangle$ and $\langle H', T, k\rangle$ are equivalent instances of HGI since a solution S of $\langle H', T, k\rangle$ cannot include any vertex in $\{s^*_v \mid v \in V\}$. Clearly, H' has maximum degree 5. To see that H' has bandwidth 4, consider the ordering that places the vertices along the path on $\{s^*_v \mid v \in V\}$, while the vertices of C_v are placed right after s^*_v. This ordering has bandwidth 4. See Figure 4. ◀

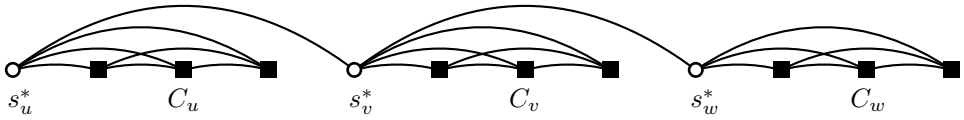


Figure 4 The embedding of H' in Corollary 3.4.

4 Fixed-parameter tractability with k plus structural parameters

4.1 Modular-width

Let $G = (V, E)$ be a graph. A set $M \subseteq V$ is a *module* of G if $N_G(u) \setminus M = N_G(v) \setminus M$ holds for all $u, v \in M$. In other words, each vertex $w \notin M$ is adjacent to all or no vertices of M. Note that if M and M' are two disjoint modules of G, then either there are all or no possible edges between them. Two modules M and M' are *adjacent* if there are all possible edges between them, and *nonadjacent* otherwise.

The *modular-width* of $G = (V, E)$, denoted $\mathsf{mw}(G)$, is defined as the minimum integer μ that satisfies the following recursive condition:

- $|V| \le \mu$; or
- V can be partitioned into modules $M_1, \ldots, M_{\mu'}$ of G such that $2 \le \mu' \le \mu$ and $\mathsf{mw}(G[M_i]) \le \mu$ for each $i \in [\mu']$.

▶ **Theorem 4.1.** *Hitting Geodesic Intervals is fixed-parameter tractable parameterized by solution size k plus modular-width.*

Proof. Let $\langle G = (V, E), T, k\rangle$ be an instance of HGI. Observe that the vertex set of each connected component C of G is a module of G, and thus $\mathsf{mw}(C) \leq \mathsf{mw}(G)$. Hence, we may assume that G is connected; otherwise we solve each connected component of G independently for all $k' \leq k$ and combine the answers for them.

Let $M_1, \ldots, M_p$ be a partition of V into modules of G with $2 \leq p \leq \mathsf{mw}(G)$. It is known that such a partition can be computed in linear time [24]. Note that, although the definition of modular-width requires a complete recursive structure, our algorithm described below only needs a partition at the very first level.

For each $\{u, v\} \in T$, we take a set $J(u, v)$ and initialize it by $I_G[u, v]$. Now we set $\mathcal{I} = \{J(u, v) \mid \{u, v\} \in T\}$. Our goal is to find a hitting set S of $\mathcal{I}$ with size at most k. To this end, we make two phases of branching that make the rest of the problem almost trivially polynomial-time solvable. Although we do not explicitly state the trivial termination conditions in each step, we stop and reject a branch if it results in $k < 0$ or $\emptyset \in \mathcal{I}$.

The first branching. We first branch on the 2^p candidates of the set of modules that S intersects, which we call $\mathcal{M}_S$. We assume that S is chosen in such a way that $|\mathcal{M}_S|$ is maximized. According to $\mathcal{M}_S$, we update $\mathcal{I}$ as follows.

- Add each $M_i \in \mathcal{M}_S$ into $\mathcal{I}$.
- For each $\{u, v\} \in T$, if there is $M_i \in \mathcal{M}_S$ with $M_i \subseteq J(u, v)$, then remove $J(u, v)$ from $\mathcal{I}$.
- For each $\{u, v\} \in T$ and $M_i \notin \mathcal{M}_S$, update $J(u, v)$ as $J(u, v) \coloneqq J(u, v) \setminus M_i$.

We now see that, for every $\{u, v\} \in T$, if u and v belong to different modules, say M_i and M_j, then $J(u, v) \subseteq \{u, v\}$ after the update above. Observe that $I_G[u, v] = \{u, v\} \cup \bigcup_{M \in \mathcal{M}} M$ for some $\mathcal{M} \subseteq \{M_1, \ldots, M_p\} \setminus \{M_i, M_j\}$. Since $J(u, v) \in \mathcal{I}$, we have $\mathcal{M} \cap \mathcal{M}_S = \emptyset$, and thus $J(u, v) \subseteq \{u, v\}$.

The second branching. Let $\mathcal{I}'$ be the subset of $\mathcal{I}$ consisting of the sets $J(u, v)$ such that $\{u, v\} \in E$ or u and v belong to different modules. Observe that every hitting set S of $\mathcal{I}$ with size at most k contains a *minimal* hitting set S' of $\mathcal{I}'$ with size at most k. We list all possible candidates for S' and branch on them. Since each member of $\mathcal{I}'$ has size at most 2, we can use the algorithm of Damaschke [9] for enumerating all minimal vertex covers of size at most k in $O(|\mathcal{I}'| + k^2 2^k)$ time. It gives us all (at most 2^k) minimal hitting sets of $\mathcal{I}'$ with size at most k (or answers that there is no such a set). After we pick one candidate S', we decrease k by $|S'|$ and update $\mathcal{I}$ by removing all elements hit by S'. In particular, we remove all $J(u, v) \in \mathcal{I}'$ from $\mathcal{I}$.

Solving the reduced instance in polynomial time. We first show that, at this point, each member of $\mathcal{I}$ is a subset of some module M_i. This is trivial for the members of $\mathcal{M}_S$. For $J(u, v) \in \mathcal{I}$, we have $\{u, v\} \notin E$ and there is a module M_i such that $u, v \in M_i$. This implies that $\mathrm{dist}_G(u, v) = 2$ since M_i has an adjacent module by the connectivity of G. Thus, $I_G[u, v]$ consists of the modules adjacent to M_i and a subset of M_i. Since $J(u, v) \in \mathcal{I}$ now, the adjacent modules do not belong to $\mathcal{M}_S$. This implies that $J(u, v) \subseteq M_i$.

For $i \in [p]$, let $\mathcal{I}_i$ be the (possibly empty) subset of $\mathcal{I}$ consisting of the members that are subsets of M_i. Since $\mathcal{I}_1, \ldots, \mathcal{I}_p$ form a partition of $\mathcal{I}$ with disjoint universes, we can handle them separately. That is, if we know a minimum hitting set S_i of $\mathcal{I}_i$ for each $i \in [p]$, then we can set $S = S' \cup \bigcup_{i \in [p]} S_i$.

We now claim that, under the assumption that $\mathcal{M}_S$ and S' are correct ones, each nonempty $\mathcal{I}_i$ admits a hitting set of size 1. Suppose to the contrary that a minimum hitting set S_i of $\mathcal{I}_i$ has size at least 2 for some $i \in [p]$. Let M_j be a module adjacent to M_i, and let $v_i \in S_i$ and $v_j \in M_j$. Let $S'_i = (S_i \setminus \{v_i\}) \cup \{v_j\}$ and $R = (S \setminus S_i) \cup S'_i$. Observe that S'_i hits $\mathcal{I}_i$: it contains at least one vertex of M_i, which hits M_i; and $v_j \in M_j$ hits all $J(u, v) \in \mathcal{I}_i$ since $\mathrm{dist}_G(u, v) = 2$. This implies that R interrupts all pairs in T as S does. However, this contradicts the assumption on S since R intersects one more module (M_j) than S. Thus, we can conclude that at least one of $\mathcal{M}_S$ and S' is not correct.

Given the observation above, we can solve the rest of the current branch in polynomial time as follow. If there are at most k indices $i \in [p]$ such that $\mathcal{I}_i$ is nonempty and each of them admits a hitting set of size 1, then return yes. Otherwise, we reject the current branch.

Total running time. We first branch on 2^p cases. In each case, we take $O^*(2^k)$ time to enumerate at most 2^k candidates for the next branch. We branch on these candidates and take polynomial time in each of them. In total, the running time is $O^*(2^{p+k})$. ◀

4.2 Vertex integrity and vertex multiway cut

To show that WHGI is fixed-parameter tractable parameterized by k plus vertex integrity and by the minimum vertex multiway-cut size of terminal vertices $V(T)$, we first give a single algorithm that applies to a more general setting that includes both cases under the assumption that a certain kind of a separator is given as part of the input. We then present simple observations of how the requirement that the separator is given as part of the input can be dropped for our cases.

4.2.1 The main algorithm

Let us first give an overview of the main algorithm. It can be seen as a generalization of the one parameterized by vertex cover number due to Aravind and Saxena [2]. Instead of a vertex cover, we use a small set $Z \subseteq V$ such that each connected component of $G - Z$ contains a small number of terminal vertices. Given Z, we guess how our solution S interrupts pairs in $\binom{Z}{2}$. To reflect the guess, we then update the family $\mathcal{I}$ of vertex sets to be hit by the solution (which was originally the family of geodesic intervals for T), in an appropriate way. We show that, after this update, each member of $\mathcal{I}$ is either a geodesic interval between two vertices in Z or a vertex set intersecting at most two connected components of $G - Z$. We then show that although each connected component of $G - Z$ may have unbounded size, we can focus on a bounded number of "representatives" in each connected component. This allows us to shrink the members of $\mathcal{I}$ and then apply the sunflower lemma to a subfamily of $\mathcal{I}$. As a result, we get an upper bound of $|\mathcal{I}|$, and thus the rest becomes easy to solve.

▶ **Theorem 4.2.** *Consider a special case of WHGI where, in addition to $\langle G = (V, E), T, k\rangle$, we are given a set $Z \subseteq V$ such that $|Z| \leq p$ and each connected component of $G - Z$ contains at most q terminal vertices. There is a fixed-parameter algorithm for this problem parameterized by $k + p + q$.*

Proof. For simplicity, we assume that Z does not include any terminal vertex. This assumption can be justified by the following simple modification: if a vertex $z \in Z$ is involved in terminal pairs, then add a new vertex z' adjacent only to z and replace all terminal pairs $\{z, x\}$ with $\{z', x\}$; this modification is safe as any path to z' must go through z. We also

assume that our solution does not use any vertex in Z. For this assumption, we guess from at most 2^p candidates the subset Z' of Z that actually is contained in the solution, remove the terminal pairs interrupted by Z', and reduce k by $|Z'|$.

Let $u, v \in V \setminus Z$ and C_u and C_v be the connected components of $G - Z$ that contain u and v, respectively. Note that C_u and C_v are not necessarily different. We now show that

$$I_G[u,v] \setminus \left(Z \cup \bigcup_{\{z,z'\} \in \binom{Z}{2}} I_G[z,z']\right) \subseteq V(C_u) \cup V(C_v). \tag{1}$$

Let P be a shortest u–v path. To show (1), it suffices to show that either $V(P) \subseteq V(C_u) \cup V(C_v)$ or $V(P) \setminus V(P') \subseteq V(C_u) \cup V(C_v)$ for some shortest path P' connecting (possibly the same) vertices $z, z' \in Z$. If P does not visit any vertex in Z, then $V(P) \subseteq V(C_u)$ holds. Assume that P visits some vertices in Z. Let P_Z be the maximal subpath of P such that the first and last vertices belong to Z. Note that P_Z is a shortest path. Let P_u and P_v be the paths containing u and v, respectively, obtained from P by removing $V(P_Z)$. Since Z separates C_u from the rest of the graph, $V(P_u) \subseteq V(C_u)$ holds. By the same argument, we can see that $V(P_v) \subseteq V(C_v)$. Hence, $V(P) \setminus V(P_Z) \subseteq V(C_u) \cup V(C_v)$ holds. This implies (1).

Branching. For each $\{u, v\} \in T$, we take a set $J(u, v)$ and initialize it by $I_G[u, v] \setminus Z$. (Recall that we already guessed and preprocessed the intersection of Z and the solution.) Now we set $\mathcal{I} = \{J(u, v) \mid \{u, v\} \in T\}$. We branch on the $2^{\binom{|Z|}{2}}$ candidates of the subset of $\binom{Z}{2}$ that S interrupts, which we call Y. That is, we assume that we are looking for a solution S such that $S \cap I_G[u, v] \neq \emptyset$ for each $\{u, v\} \in Y$, and $S \cap I_G[u, v] = \emptyset$ for each $\{u, v\} \in \binom{Z}{2} \setminus Y$. Thus, we update $\mathcal{I}$ as follows.

- For each $\{u, v\} \in Y$, let $J(u, v) = I_G[u, v] \setminus Z$ and add $J(u, v)$ into $\mathcal{I}$.
- For each $\{u, v\} \in T$ and $\{u', v'\} \in Y$, if $J(u', v') \subseteq J(u, v)$, then remove $J(u, v)$ from $\mathcal{I}$.
- For each $\{u, v\} \in T$ and $\{u', v'\} \in \binom{Z}{2} \setminus Y$, if $J(u', v') \subseteq J(u, v)$, then update $J(u, v)$ as $J(u, v) \coloneqq J(u, v) \setminus J(u', v')$.

Let $\mathcal{I}_T = \{J(u, v) \in \mathcal{I} \mid \{u, v\} \in T\}$ and $\mathcal{I}_Y = \{J(u, v) \in \mathcal{I} \mid \{u, v\} \in Y\}$. The condition (1) implies that, after the update above, $J(u, v) \subseteq V(C_u) \cup V(C_v)$ holds for each $J(u, v) \in \mathcal{I}_T$.

Reducing the size of each set in $\mathcal{I}_T$. Two vertices u and v are *T-equivalent* if u and v interrupt the same set of pairs in T. In other words, u and v are T-equivalent if, for each $\{x, y\} \in T$, either $\{u, v\} \subseteq I_G[x, y]$ or $\{u, v\} \cap I_G[x, y] = \emptyset$ holds. Observe that the T-equivalence among vertices is an equivalence relation. Every inclusion-wise minimal solution for $\langle G, T, k\rangle$ includes at most one vertex of a T-equivalence class, and if it includes one vertex of a T-equivalence class, then taking any of them is equivalent. The partition of the vertices to the T-equivalence classes can be computed in polynomial time by first computing all-pairs shortest distance in G, and then iteratively refining the partition of the vertices with respect to the interruption relation to each pair $\{u, v\} \in T$.

Let V' be a subset of V constructed by picking one vertex from each T-equivalence class. Using V', we further update $\mathcal{I}$ by compressing each $J(u, v) \in \mathcal{I}$ as $J(u, v) \coloneqq J(u, v) \cap V'$. We now show that this update allows us to upper-bound the size of each $J(u, v) \in \mathcal{I}_T$ by a function depending only on $p + q$. To this end, we prove the following claim.

▷ Claim 4.3. Let C be a connected component of $G - Z$. If two vertices $x, y \in V(C)$ are $\binom{Z \cup (V(T) \cap V(C))}{2}$-equivalent, then they are T-equivalent.

Proof. It suffices to show that if one of x and y interrupts a pair $\{a, b\} \in T$, then the other also interrupts $\{a, b\}$. By symmetry, we may assume that x interrupts $\{a, b\}$.

Let P_{ax} be a shortest a–x path and P_{xb} a shortest x–b path. Since x interrupts $\{a, b\}$, the concatenation $P^{(x)}_{ab} \coloneqq P_{ax}P_{xb}$ is a shortest a–b path. Now we pick two vertices $a' \in V(P_{ax})$ and $b' \in V(P_{xb})$ as follows: we set $a' = a$ if $a \in V(C)$; otherwise, we set a' to an arbitrary vertex in $V(P_{ax}) \cap Z$ (such a vertex exists as Z separates C from the rest of the graph); analogously, we set $b' = b$ if $b \in V(C)$; otherwise, we set b' to an arbitrary vertex in $V(P_{xb}) \cap Z$. By the definition, $\{a', b'\} \subseteq Z \cup (V(T) \cap V(C))$ holds.

Let $P^{(x)}_{a'b'}$ be the subpath of $P^{(x)}_{ab}$ from a' to b'. Since $P^{(x)}_{a'b'}$ is a shortest a'–b' path containing x, x interrupts $\{a', b'\}$, and thus so does y by their $\binom{Z \cup (V(T) \cap V(C))}{2}$-equivalence. Thus, there is a shortest a'–b' path $P^{(y)}_{a'b'}$ that passes through y. This implies that we can obtain a shortest a–b path that visits y by replacing $P^{(x)}_{a'b'}$ in $P^{(x)}_{ab}$ with $P^{(y)}_{a'b'}$. Hence, y interrupts $\{a, b\}$. ◁

For every connected component C of $G - Z$, the assumption on Z implies that $|Z \cup (V(T) \cap V(C))| \leq p + q$. Thus, Claim 4.3 implies that C intersects at most $2^{\binom{|Z \cup (V(T) \cap V(C))|}{2}} \leq 2^{\binom{p+q}{2}}$ T-equivalence classes, and thus, $|V' \cap V(C)| \leq 2^{\binom{p+q}{2}}$. This implies that $|J(u, v)| \leq 2^{\binom{p+q}{2}+1}$ for each $J(u, v) \in \mathcal{I}_T$ since $J(u, v) \subseteq V' \cap (V(C_u) \cup V(C_v))$.

Reducing the size of $\mathcal{I}$. Now we shrink the size of $\mathcal{I}_T$ by a well-known application of the sunflower lemma [14] to d-Hitting Set, a variant of Hitting Set with each set having size at most d. There is a polynomial-time algorithm based on the sunflower lemma that, given an instance $\mathcal{H}$ of d-Hitting Set, either

- answers that $\mathcal{H}$ admits no hitting set of size at most k, or
- returns an instance $\mathcal{H}'$ of d-Hitting Set with at most $k^d \cdot d!$ sets such that $\mathcal{H}'$ and $\mathcal{H}$ admit exactly the same (possibly empty) family of hitting sets of size at most k.

See [15, Section 9.1] or [8, Theorem 2.26]. By setting $d = 2^{\binom{p+q}{2}+1}$, we apply this algorithm to $\mathcal{I}_T$ and obtain $\mathcal{I}'_T$ with at most $k^d \cdot d!$ sets (or find that it is a no-instance). Finally, we set $\mathcal{I}' = \mathcal{I}'_T \cup \mathcal{I}_Y$. The discussion so far guarantees that $\mathcal{I}'$ and $\mathcal{I}$ are equivalent. Since $|\mathcal{I}'|$ $(= |\mathcal{I}'_T| + |\mathcal{I}_Y| \leq k^d \cdot d! + p^2)$ depends only on $k + p + q$, we can find a minimum hitting set of $\mathcal{I}'$ by applying the standard fixed-parameter algorithm for Hitting Set parameterized by the number of sets (see Section 1.1). ◀

4.2.2 Applications of the main algorithm

Theorem 4.2 requires a separator Z with special properties as part of the input. A natural question would be whether the original problem WHGI is fixed-parameter tractable parameterized by $k + p + q$. To this end, it suffices to show that finding Z is fixed-parameter tractable parameterized by $p + q$. Although we leave this general problem unsettled, we now observe that for our special cases finding Z is fixed-parameter tractable.

The *vertex integrity* of a graph $G = (V, E)$, denoted $\mathsf{vi}(G)$, is the minimum integer ι such that there is a set $X \subseteq V$ satisfying that $\iota = |X| + \max_{C \in \mathsf{cc}(G-X)} |V(C)|$, where $\mathsf{cc}(G - X)$ is the set of connected components of $G - X$. For an n-vertex graph G with $\mathsf{vi}(G) = \iota$, finding a set $X \subseteq$ satisfying $\iota = |X| + \max_{C \in \mathsf{cc}(G-X)} |V(C)|$ can be done in time $O(\iota^{\iota+1}n)$ [13]. Observe that such a set X satisfies the conditions of Z in Theorem 4.2 with $p = q = \iota$. Thus we get the following result as a direct corollary.

▶ **Corollary 4.4.** *Weighted Hitting Geodesic Intervals is fixed-parameter tractable parameterized by solution size k plus vertex integrity.*

For a graph $G = (V, E)$ and $U \subseteq V$, a set $X \subseteq V$ is a *vertex multiway cut* for U if each connected component of $G - X$ contains at most one vertex of U. It is known that the problem of finding a vertex multiway cut of size r is fixed-parameter tractable parameterized by r [6, 22]. A multiway cut X for $V(T)$ with size r satisfies the conditions of Z in Theorem 4.2 with $p = r$ and $q = 1$. Furthermore, X interrupts all pairs in T as no path can connect a pair in T without passing through X. Thus, we can assume that $k \leq r$ as otherwise we already have a desired solution (i.e., X). Now we apply Theorem 4.2 and get the following result.

▶ **Corollary 4.5.** *Weighted Hitting Geodesic Intervals is fixed-parameter tractable parameterized by the minimum vertex multiway-cut size of the set of terminal vertices $V(T)$.*

As a final remark to this section, we note that the problem of finding Z in Theorem 4.2 is *nonuniformly* fixed-parameter tractable parameterized by $p + q$. This follows from the discussion by Marx [22, p. 396] on the case of $q = 1$ (i.e., the vertex multiway cut problem), which directly applies to general q as well. Given $G = (V, E)$ and $R \subseteq V$, we construct an edge-colored graph $G' = (V', E')$ as follows: for each $v \in R$, add a vertex v' and an edge $\{v, v'\}$; color E in black and $\{\{v, v'\} \mid v \in R\}$ in red. Now, there is $Z \subseteq V$ such that $|Z| \leq p$ and each connected component of $G - Z$ contains at most q terminal vertices if and only if there is $Z' \subseteq V'$ such that $|Z'| \leq p$ and each connected component of $G' - Z'$ contains at most q red edges. Observe that the latter is a minor-closed property. More precisely, let $\mathcal{G}_{p,q}$ be the family of graphs with black and red edges such that each graph $H \in \mathcal{G}_{p,q}$ admits a separator $X \subseteq V(H)$ such that $|X| \leq p$ and each connected component of $H - X$ contains at most q red edges. Then, $\mathcal{G}_{p,q}$ is closed under the operation of taking minors. Thus, the edge-colored version of the graph minor theorem (see [12, Exercise 19.5.6]) implies that testing the membership to $\mathcal{G}_{p,q}$ is fixed-parameter tractable parameterized by $p + q$.

5 Parameterized intractability beyond k

Since HGI parameterized by k belongs to W[2], both subcases handled in this section also belong to W[2]. Hence, we only prove the W[2]-hardness in each case.

For a graph $G = (V, E)$ and a set T of vertex pairs, a set $X \subseteq V$ is a *vertex multicut* for T if no connected component of $G - X$ contains a pair in T. It is known that the problem of finding a vertex multicut is fixed-parameter tractable parameterized by its size [4, 23].

Observe that a vertex multiway cut for $V(T)$ is a vertex multicut for T, but not vice versa. Observe also that a vertex multicut for T interrupts all pairs in T. Therefore, the minimum vertex *multicut* size is upper-bounded by the minimum vertex *multiway-cut* size, for which WHGI is fixed-parameter tractable (Corollary 4.5), and lower-bounded by the solution size k, for which HGI is W[2]-complete [2]. Thus, the following result sharpens the border of (in)tractability in this parameter hierarchy.

▶ **Theorem 5.1.** *Hitting Geodesic Intervals is* W[2]*-complete parameterized by the minimum vertex multicut size of the terminal pairs T.*

Proof. Let $\langle U, \mathcal{F}, h\rangle$ be an instance of Hitting Set, where $U = \{u_1, \ldots, u_n\}$ and $\mathcal{F} = \{F_1, \ldots, F_m\} \subseteq 2^U$. From this instance, we construct an equivalent instance $\langle G, T, h\rangle$ of HGI with $G = (V, E)$ as follows (see Figure 5 (left)). Let $W = \{w_i \mid i \in [h + 3]\}$.[3] We set

[3] For proving Theorem 5.1, it suffices to set $|W| = h + 1$ instead of $h + 3$. We use $h + 3$ to make the proof of Theorem 5.2 simpler.

$V = \mathcal{F} \cup U \cup W$. For $u_i \in U$ and $F_j \in \mathcal{F}$, we add $\{u_i, F_j\}$ into E if and only if $u_i \in F_j$. We then add all possible edges between U and W. We set $T = \{\{F_i, w_j\} \mid i \in [m],\ j \in [h+3]\}$. We can see that W is a vertex multicut of T with size $h+3$.

Assume that $\langle U, \mathcal{F}, h\rangle$ is a yes-instance of Hitting Set. Let $S \subseteq U$ be a hitting set of $\mathcal{F}$ with $|S| \leq h$. Since S is a hitting set of $\mathcal{F}$, each $F_i \in \mathcal{F}$ includes a member of S, say u_j, and that member u_j interrupts the pair $\{F_i, w_p\}$ for each $p \in [h+3]$.

Assume that $\langle G, T, h\rangle$ is a yes-instance of HGI. We first claim that there is a solution for $\langle G, T, h\rangle$ that does not include any vertex in $\mathcal{F}$. To see this, observe that F_i interrupts all pairs $\{F_i, w_p\}$ for $p \in [h+3]$ but no others, while each $u_j \in F_i$ also interrupts them. Let S be a solution for $\langle G, T, h\rangle$ with $S \cap \mathcal{F} = \emptyset$. Since $|S| \leq h < |W|$, there is a vertex $w_p \in W \setminus S$. Since no other vertex $w_q \in W \setminus \{w_p\}$ belongs to the geodesic interval $I_G[w_p, F_i]$ for every $i \in [m]$, the set $S \setminus W$ $(= S \cap U)$ has to interrupt $\{w_p, F_i\}$ for every $i \in [m]$. This implies that, each F_i has a neighbor u_j in $S \cap U$, and thus, $S \cap U$ is a hitting set of $\mathcal{F}$. ◀

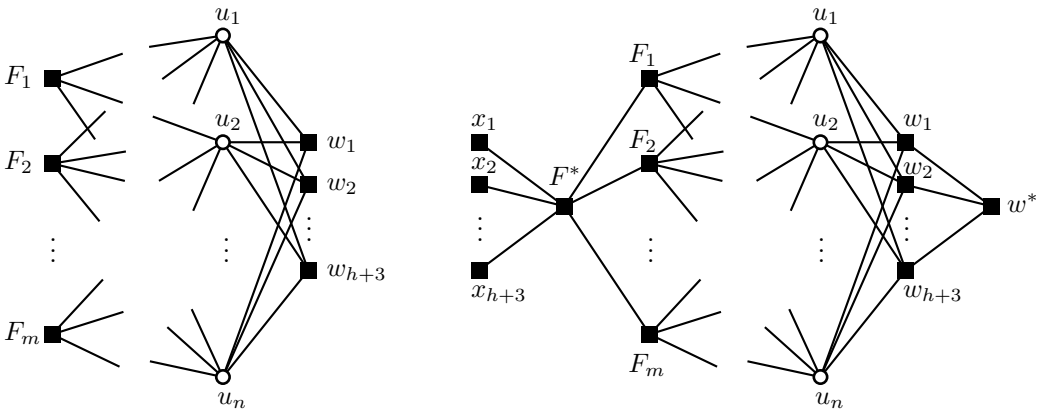


Figure 5 The graphs G (left) and H (right) in Theorems 5.1 and 5.2, respectively.

We now modify the proof of Theorem 5.1 and show that HGI is W[2]-complete parameterized by k even if the structure formed by the terminal pairs is quite limited, i.e., $T = \binom{Q}{2}$ for some $Q \subseteq V$. In other words, we consider the setting where we want to hit pairwise geodesic intervals in a given vertex set Q. As this variant could be of independent interest, we define it as a separate problem as follows.

Problem: Hitting Pairwise Geodesic Intervals (HPGI)
Input: A graph $G = (V, E)$, a set $Q \subseteq V$, and an integer k.
Question: Is there $S \subseteq V$ with $|S| \leq k$ such that $S \cap I_G[u, v] \neq \emptyset$ for each $\{u, v\} \in \binom{Q}{2}$?

▶ **Theorem 5.2.** *Hitting Pairwise Geodesic Intervals is* W[2]*-complete parameterized by solution size k.*

Proof. We first repeat the construction in the proof of Theorem 5.1 to obtain an instance $\langle G = (V, E), T, h\rangle$ of HGI from an instance $\langle U, \mathcal{F}, h\rangle$ of Hitting Set. From $\langle G, T, h\rangle$, we construct an equivalent instance $\langle H, Q, h+2\rangle$ of HPGI (see Figure 5 (right)): to obtain H from G, we add vertices F^* and w^* and a set $X = \{x_i \mid i \in [h+3]\}$ of vertices, and then add all possible edges between F^* and $X \cup \mathcal{F}$ and between w^* and W; we set $Q = V(H) \setminus U$.

Assume that $\langle G, T, h\rangle$ is a yes-instance of HGI. Let $S \subseteq U$ be a solution for $\langle G, T, h\rangle$. We set $S' = S \cup \{F^*, w^*\}$. We can see that S' interrupts all pairs in $\binom{Q}{2}$: S interrupts all pairs in $T = \{\{F_i, w_j\} \mid i \in [m],\ j \in [h+3]\}$; F^* interrupts all pairs involving a vertex in X and all pairs $\{F_i, F_j\}$; and w^* interrupts all pairs $\{w_i, w_j\}$.

Assume that $\langle H, Q, h+2\rangle$ is a yes-instance of HPGI. Let S be a solution for $\langle G, Q, h\rangle$. Observe that $F^* \in S$ since otherwise we have to take all $h+3$ vertices of X into S. Similarly, we can see that $w^* \in S$. Since neither F^* nor w^* can interrupt pairs $\{F_i, w_p\} \in T$ and $I_G[F_i, w_p] = I_H[F_i, w_p]$ for all $i \in [m]$ and $j \in [h+3]$, we can conclude that $S \setminus \{F^*, w^*\}$ interrupts all pairs in T. ◀

6 Concluding remarks

In this paper, we showed that Hitting Geodesic Intervals is intractable even on graphs with highly restricted structures, while combinations of the solution size k and some structural graph parameters make the problem fixed-parameter tractable (see Figure 1). As Figure 1 (right) shows, when k is part of the parameter, many cases remain unsettled. It would be interesting to further investigate these cases.

In one of our fixed-parameter algorithms (Theorem 4.2), we have a restriction that a certain kind of a separator is given as part of the input, and then we discussed that for our target cases it is actually not a restriction (Corollaries 4.4 and 4.5). We wonder if requiring such a separator in the input is not a restriction in general. More precisely, we would like to ask whether the following problem is fixed-parameter tractable parameterized by $p+q$.

Input: A graph $G = (V, E)$, a set of terminal vertices $R \subseteq V$, and integers p and q.
Question: Is there $Z \subseteq V$ with $|Z| \leq p$ such that each connected component C of $G - Z$ contains at most q terminal vertices (i.e., $|V(C) \cap R| \leq q$)?

Note that, as we discussed in the last paragraph of Section 4.2.2, the graph minor theorem gives a nonuniform fixed-parameter algorithm parameterized by $p+q$ for this problem, while we are looking for a uniform one.

References

1 Pankaj K. Agarwal, Sariel Har-Peled, Rahul Raychaudhury, and Stavros Sintos. Fast approximation algorithms for piercing boxes by points. In *SODA 2024*, pages 4892–4908, 2024. doi:10.1137/1.9781611977912.174.

2 N. R. Aravind and Roopam Saxena. The parameterized complexity of terminal monitoring set. In *WALCOM 2024*, volume 14549 of *Lecture Notes in Computer Science*, pages 76–90, 2024. doi:10.1007/978-981-97-0566-5_7.

3 Mustafa Atici. Computational complexity of geodetic set. *Int. J. Comput. Math.*, 79(5):587–591, 2002. doi:10.1080/00207160210954.

4 Nicolas Bousquet, Jean Daligault, and Stéphan Thomassé. Multicut is FPT. *SIAM J. Comput.*, 47(1):166–207, 2018. doi:10.1137/140961808.

5 Boštjan Brešar, Matjaž Kovše, and Aleksandra Tepeh. Geodetic sets in graphs. In *Structural Analysis of Complex Networks*, pages 197–218. Birkhäuser, 2011. doi:10.1007/978-0-8176-4789-6_8.

6 Jianer Chen, Yang Liu, and Songjian Lu. An improved parameterized algorithm for the minimum node multiway cut problem. *Algorithmica*, 55(1):1–13, 2009. doi:10.1007/S00453-007-9130-6.

7 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 4th edition, 2022.

8 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

9 Peter Damaschke. Parameterized enumeration, transversals, and imperfect phylogeny reconstruction. *Theor. Comput. Sci.*, 351(3):337–350, 2006. doi:10.1016/J.TCS.2005.10.004.

10 Aaron L. Douthat and Man C. Kong. Computing geodetic bases of chordal and split graphs. *J. Comb. Math. Comb. Comput.*, 22:67–77, 1996.

11 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer, 1999. doi:10.1007/978-1-4612-0515-9.

12 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

13 Pål Grønås Drange, Markus S. Dregi, and Pim van 't Hof. On the computational complexity of vertex integrity and component order connectivity. *Algorithmica*, 76(4):1181–1202, 2016. doi:10.1007/s00453-016-0127-x.

14 Paul Erdős and Richard Rado. Intersection theorems for systems of sets. *J. London Math. Soc.*, 35(1):85–90, 1960. doi:10.1112/jlms/s1-35.1.85.

15 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer, 2006. doi:10.1007/3-540-29953-X.

16 Robert J. Fowler, Mike Paterson, and Steven L. Tanimoto. Optimal packing and covering in the plane are NP-complete. *Inf. Process. Lett.*, 12(3):133–137, 1981. doi:10.1016/0020-0190(81)90111-3.

17 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

18 Frank Harary, Emmanuel Loukakis, and Constantine Tsouros. The geodetic number of a graph. *Math. Comput. Model.*, 17(11):89–95, 1993. doi:10.1016/0895-7177(93)90259-2.

19 Bart M. P. Jansen. On structural parameterizations of hitting set: Hitting paths in graphs using 2-SAT. *J. Graph Algorithms Appl.*, 21(2):219–243, 2017. doi:10.7155/JGAA.00413.

20 Bart M. P. Jansen and Céline M. F. Swennenhuis. Steiner tree parameterized by multiway cut and even less. In *ESA 2024*, volume 308 of *LIPIcs*, pages 76:1–76:16, 2024. doi:10.4230/LIPICS.ESA.2024.76.

21 Leon Kellerhals and Tomohiro Koana. Parameterized complexity of geodetic set. *J. Graph Algorithms Appl.*, 26(4):401–419, 2022. doi:10.7155/JGAA.00601.

22 Dániel Marx. Parameterized graph separation problems. *Theor. Comput. Sci.*, 351(3):394–406, 2006. doi:10.1016/J.TCS.2005.10.007.

23 Dániel Marx and Igor Razgon. Fixed-parameter tractability of multicut parameterized by the size of the cutset. *SIAM J. Comput.*, 43(2):355–388, 2014. doi:10.1137/110855247.

24 Ross M. McConnell and Jeremy P. Spinrad. Modular decomposition and transitive orientation. *Discrete Mathematics*, 201(1-3):189–241, 1999. doi:10.1016/S0012-365X(98)00319-7.

25 Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006. doi:10.1093/ACPROF:OSO/9780198566076.001.0001.

26 Johannes C. B. Schröder. Comparing graph parameters. Bachelor thesis, Technische Universität Berlin, 2019. URL: https://fpt.akt.tu-berlin.de/publications/theses/BA-Schr%C3%B6der.pdf.

27 Manuel Sorge and Mathias Weller. The graph parameter hierarchy, 2019. URL: https://manyu.pro/assets/parameter-hierarchy.pdf.

28 Prafullkumar Tale. Geodetic set on graphs of constant pathwidth and feedback vertex set number. *CoRR*, abs/2504.17862, 2025. doi:10.48550/arXiv.2504.17862.

29 Duc Long Tran. Expanding the graph parameter hierarchy. Bachelor thesis, Technische Universität Berlin, 2022. URL: https://fpt.akt.tu-berlin.de/publications/theses/BA-Duc-Long-Tran.pdf.

A Graph Width Perspective on Partially Ordered Hamiltonian Paths and Cycles II

Vertex and Edge Deletion Numbers

Jesse Beisegel
Institute of Mathematics, Brandenburg University of Technology, Cottbus, Germany

Katharina Klost
Institute of Computer Science, Freie Universität Berlin, Germany

Kristin Knorr
Institute of Computer Science, Freie Universität Berlin, Germany

Fabienne Ratajczak
Institute of Mathematics, Brandenburg University of Technology, Cottbus, Germany

Robert Scheffler
Institute of Mathematics, Brandenburg University of Technology, Cottbus, Germany

Abstract

We consider the problem of finding a Hamiltonian path or cycle with precedence constraints in the form of a partial order on the vertex set. We study the complexity for graph width parameters for which the ordinary problems Hamiltonian Path and Hamiltonian Cycle are in FPT. In particular, we focus on parameters that describe how many vertices and edges have to be deleted to become a member of a certain graph class. We show that the problems are W[1]-hard for such restricted cases as vertex distance to path and vertex distance to clique. We complement these results by showing that the problems can be solved in XP time for vertex distance to outerplanar and vertex distance to block. Furthermore, we present some FPT algorithms, e.g., for edge distance to block. Additionally, we prove para-NP-hardness when considered with the edge clique cover number.

2012 ACM Subject Classification Mathematics of computing → Paths and connectivity problems; Mathematics of computing → Graph algorithms; Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Problems, reductions and completeness

Keywords and phrases Hamiltonian path, Hamiltonian cycle, partial order, graph width parameter, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.30

Related Version *Full Version*: `https://arxiv.org/abs/2510.08378` [6]

Funding *Fabienne Ratajczak*: The research was funded by the *Federal Ministry for Digital and Transport Germany* through the research project *MoVeToLausitz* (grant number 19FS2032C).

1 Introduction

Hamiltonian Paths and Cycles with Precedence. For some applications of the well-known Traveling Salesman Problem (TSP) it is necessary to add additional precedence constraints to the vertices which ensure that some vertices are visited before others in a tour. Examples are *Pick-up and Delivery* Problems [42, 43] or the *Dial-a-Ride* problem [45], where goods or people have to be picked up before they can be brought to their destination.

The generalization of TSP with this type of constraint is known as Traveling Salesman Problem with Precedence Constraints (TSP-PC) and has been studied, e.g., in [1, 9]. If we are looking for a path instead of a cycle, this is known as the Sequential Ordering Problem (SOP) or the Minimum Setup Scheduling Problem and has been studied, e.g., in [2, 11, 22, 23].

© Jesse Beisegel, Katharina Klost, Kristin Knorr, Fabienne Ratajczak, and Robert Scheffler;
licensed under Creative Commons License CC-BY 4.0

As TSP is already NP-complete without the precedence constraints, research on these problems has mainly focused on the practical applications and solution methods are restricted to heuristic algorithms and integer programming. When moving to the structurally different Hamiltonian path (cycle) problem, only very restricted precedence constraints have been studied, e.g., where one or both endpoints of the Hamiltonian path are fixed. For these problems, polynomial-time algorithms have been presented for several graph classes including (proper) interval graphs [3, 4, 38, 39] and rectangular grid graphs [33].

In [7], Beisegel et al. introduce the Partially Ordered Hamiltonian Path Problem (POHPP), where we are given a graph together with a partial order and search for a Hamiltonian path that is a linear extension of the partial order. Additionally, they introduced the edge-weighted variant Minimum Partially Ordered Hamiltonian Path Problem (MinPOHPP). The authors show that POHPP is already NP-hard for complete bipartite graphs and complete split graphs – graph classes where Hamiltonian Path is trivial. They also show that POHPP is W[1]-hard when parameterized by the *width* of the partial order, i.e., the largest number of pairwise incomparable elements. Furthermore, they show that the XP algorithm for that parameterization presented in the 1980s by Coulbourn and Pulleyblank [11] is asymptotically optimal – assuming the Exponential Time Hypothesis (ETH). They improve the algorithm to FPT time if the problem is parameterized by the partial order's *distance to linear order*. Finally, the authors present a polynomial-time algorithm for MinPOHPP on outerplanar graphs.

Graph Width Parameters and Hamiltonicity. In [7], Beisegel et al. imply that their result for outerplanar graphs might be extended to series parallel graphs, otherwise known as the graphs of treewidth two. Furthermore, they suggest further research on the complexity of (Min)POHPP for graphs of bounded treewidth as well as other graph width parameters. Just as for specific graph classes, the prerequisite to solving (Min)POHPP for some graph parameter is the tractability of the base problems Hamiltonian Path and Hamiltonian Cycle. These belong to the most famous NP-complete graph problems and, hence, have been studied for a wide range of graph classes and graph width parameters.[1]

One of the first results for width parameters were polynomial-time algorithms for Hamiltonian Cycle and TSP on graphs of bounded *bandwidth* [31, 40]. Since both Hamiltonian Cycle and Hamiltonian Path are expressible in MSO_2, Courcelle's theorem [12, 13] implies FPT algorithms for these problems when parameterized by *treewidth*. The running time bounds depending on the *treewidth* k given by Courcelle's theorem are quite bad. However, there are also FPT algorithms with single exponential dependency on k [10, 14, 15]. Probably, these results cannot be transferred to *cliquewidth*: In contrast to MSO_2, both Hamiltonian Cycle and Hamiltonian Path are not expressible in MSO_1 [21, Corollary 5.3.5]. In fact, Hamiltonian Cycle is W[1]-hard when parameterized by *cliquewidth* [27] and – unless $\mathsf{FPT} = \mathsf{W}[1]$ – it is not solvable in FPT time. Furthermore, it was shown that – unless the ETH fails – the problem cannot be solved in $f(k)n^{o(k)}$ time where k is the *cliquewidth* and n is the number of vertices [28]. This lower bound is matched by an $f(k)n^{\mathcal{O}(k)}$ algorithm given by Bergougnoux et al. [8]. XP algorithms for *cliquewidth* with worse exponents have been presented earlier by Wanke [48] (for the equivalent NLC-width) and by Espelage et al. [24].

As FPT algorithms for Hamiltonian Path and Hamiltonian Cycle parameterized by *cliquewidth* are highly unlikely, the research has focused on upper bounds of *cliquewidth*. FPT algorithms were presented for *neighborhood diversity* [37], *distance to cluster* [19, 34], *modular*

[1] In general, Hamiltonian Path seems to lead a shadowy existence since many positive and negative algorithmic results are only given for its more popular sibling Hamiltonian Cycle.

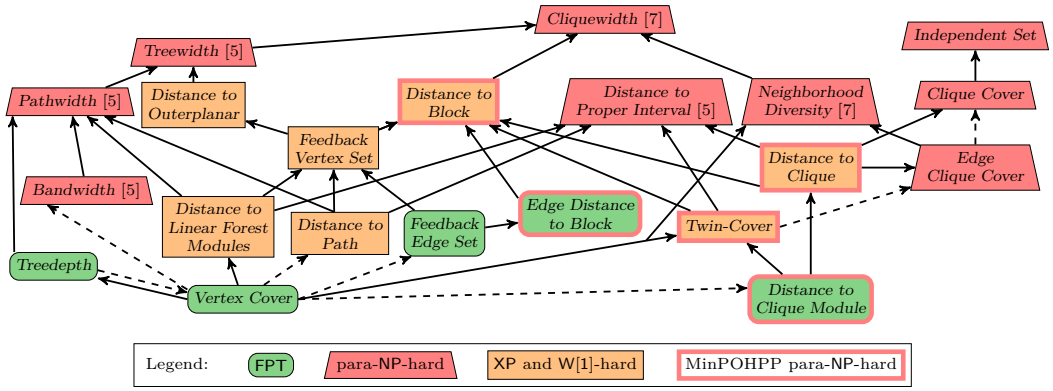


Figure 1 Diagram illustrating the complexity results for (Min)POHPP for different graph width parameters. A directed solid edge from parameter P to parameter Q means that a bounded value of P implies a bounded value for Q. A directed dashed edge implies that this relation does not hold in general but for traceable graphs, i.e., graphs having a Hamiltonian path. If a directed solid path from P to Q is missing, then parameter Q is unbounded for the graphs of bounded P. The same holds for the traceable graphs if there is also no path using dashed edges.

width [29] and *split matching width* [46, 47]. Besides this, parameterized algorithms have also been given for graph width parameters incomparable to *cliquewidth* and *treewidth*. Examples are FPT algorithms for *distance to proper interval graphs* [32] and for two parameters that describe the distance to Dirac's condition on the existence of Hamiltonian paths [35]. Most recently, FPT algorithms for the *independence number* have been presented [26].

The parameters *treewidth*, *pathwidth*, and *bandwidth* in the context of (Min)POHPP have been studied in the companion paper [5], where we showed that POHPP is NP-hard for small constants. Furthermore, it follows directly from the NP-completeness of POHPP on complete bipartite graphs that it is also NP-complete on graphs of *cliquewidth* 2, *neighborhood diversity* 2, *modular width* 2 and *split matching width* 1. As implied before, the next likely candidates to yield FPT or XP algorithms are the parameters that consider the smallest number of vertices or edges that have to be removed such that the resulting graph is in some given graph class. We refer to these parameters as *(vertex) distance to* $\mathcal{G}$ and *edge distance to* $\mathcal{G}$ where $\mathcal{G}$ is the respective graph class.[2] Here, we will also consider a special variant of these distance parameters that is motivated by the parameter *twin-cover number* introduced by Ganian [30]. We say a graph has *distance to* $\mathcal{G}$ *module(s)*[3] k if there is a set $W \subseteq V(G)$ of k vertices such that $G - W$ is in $\mathcal{G}$ and every component of $G - W$ forms a module in G. Using this terminology, the *twin-cover number* is equal to the *distance to cluster modules*.

Our Contribution. We study the complexity of (Min)POHPP and its cycle variant for graph width parameters where Hamiltonian Path and Hamiltonian Cycle can be solved in FPT time (see Figure 1 for an overview). We will show that the computational complexity of the path and cycle problems are essentially equivalent (see Observations 2.1 and 2.2) and therefore all proofs will focus on the path version.

[2] Alternative terms for these parameters are $\mathcal{G}$ *vertex deletion number* and $\mathcal{G}$ *edge deletion number*.
[3] If the class $\mathcal{G}$ contains only connected graphs, then we use *distance to* $\mathcal{G}$ *module* otherwise we use *distance to* $\mathcal{G}$ *modules*.

Section 3 is dedicated to sparse parameters, i.e., graph parameters that are unbounded for complete graphs. We show that POHPP is W[1]-hard for *distance to path* and *distance to linear forest modules*. We present simple arguments why MinPOHPP can be solved in FPT time for *feedback edge set number* and *treedepth*. Furthermore, we present an XP algorithm for *distance to outerplanar*. We also give an FPT algorithm for plane graphs parameterized by the number of vertices that lie in the interior of the outer face.

In Section 4, we consider graph width parameters that are bounded for cliques. We show that POHPP is NP-complete for graphs of *edge clique cover number* 3 and *clique cover number* 2. Furthermore, we prove W[1]-hardness for *distance to clique* and *distance to cluster modules* aka *twin cover number*. On the positive side, we present an XP algorithm for POHPP when parameterized by *distance to block* and FPT algorithms when parameterized by *edge distance to block* or *distance to clique module*.

Due to space constraints, we omit some of the proofs and sketch the others. The detailed proofs can be found in the full version [6].

2 Preliminaries

General Notation and Partial Orders. For $n \in \mathbb{N}$, the notation $[n]$ refers to the set $\{i \in \mathbb{N} \mid 1 \leq i \leq n\}$. A *partial order* π on a set X is a reflexive, antisymmetric and transitive relation on X. The tuple (X, π) is then called a *partially ordered set*. We also denote $(x, y) \in \pi$ by $x \prec_\pi y$ if $x \neq y$. If it is clear which partial order is meant, then we sometimes omit the index. A *minimal element* of a partial order π on X is an element $x \in X$ for which there is no element $y \in X$ with $y \prec_\pi x$.

A *linear ordering* of a finite set X is a bijection $\sigma : X \to \{1, 2, \ldots, |X|\}$. We will often refer to linear orderings simply as orderings. Furthermore, we will denote an ordering by a tuple $(x_1, \ldots, x_n)$ which means that $\sigma(x_i) = i$. Given two elements x and y in X, we say that x is *to the left* (resp. *to the right*) of y if $\sigma(x) < \sigma(y)$ (resp. $\sigma(x) > \sigma(y)$) and we denote this by $x \prec_\sigma y$ (resp. $x \succ_\sigma y$). A *linear extension* of a partial order π is a linear ordering σ of X that fulfills all conditions of π, i.e., if $x \prec_\pi y$, then $x \prec_\sigma y$.

Graphs and Graph Classes. All the graphs considered here are finite. For the standard notation of graphs we refer to the book of Diestel [18].

For a subgraph H of G, we say that two vertices $v, w \in V(H)$ are *H-neighbors* if $vw \in E(H)$. A vertex v of a connected graph G is a *cut vertex* if $G - v$ is not connected. If G does not contain a cut vertex, then G is *2-connected*. A *block* of a graph is an inclusion-maximal 2-connected induced subgraph. The *block-cut tree* $\mathcal{T}$ of G is the bipartite graph that contains a vertex for every cut vertex of G and a vertex for every block of G and the vertex of block B is adjacent to the vertex of a cut vertex v in $\mathcal{T}$ if B contains v.

A graph is a *linear forest* if its connected components are paths. A graph is a *cluster graph* if its components are cliques. A graph is a *block graph* if its blocks are cliques.

A graph is *planar* if it has a crossing-free embedding in the plane, and together with this embedding it is called a *plane graph*. For a plane graph G we call the regions of $\mathbb{R}^2 \setminus G$ the *faces* of G. Every plane graph has exactly one unbounded face which is called the *outer face*. Note that we will use the face and the closed walk of G that forms the border of that face interchangeably. A graph is called *outerplanar* if it has a crossing-free embedding such that all of the vertices belong to the outer face and such an embedding is also called *outerplanar*.

Graph Width Parameters. An *(edge) clique cover* of a graph G is a set of cliques of G that cover all vertices (all edges) of the graph. The *(edge) clique cover number* of G is the minimal size of an (edge) clique cover of G.

Let $\mathcal{G}$ be a graph class. The *distance to* $\mathcal{G}$ of a graph G is the minimal number of vertices that need to be removed from G so that the resulting graph belongs to $\mathcal{G}$. Similarly, the *edge distance to* $\mathcal{G}$ of a graph G is the minimal number of edges that need to be removed from G so that the resulting graph belongs to $\mathcal{G}$. A non-trivial *module* of a graph is a strict subset $M \subsetneq V(G)$ such that for all vertices $u, v \in M$ it holds that every neighbor of u outside of M is also a neighbor of v. The *distance to* $\mathcal{G}$ *module(s)* is the minimal number of vertices that need to be removed from G so that the resulting graph is in $\mathcal{G}$ and every component of the resulting graph is a non-trivial module in G. Several of the distance parameters have their own name. *Vertex cover number* is equivalent to *distance to edgeless*, *feedback vertex (edge) set number* is equivalent to *(edge) distance to tree* and *twin-cover number* [30] is equivalent to *distance to cluster modules*.

Hamiltonian Paths and Cycles. A *Hamiltonian path (cycle)* of a graph G is a path (cycle) that contains all the vertices of G. A graph is *traceable* if it has a Hamiltonian path and *Hamiltonian* if it has a Hamiltonian cycle. Here, we only consider *ordered* Hamiltonian paths, i.e., one of the two possible orderings of the path is fixed. Given a partial order π on a graph's vertex set, an ordered Hamiltonian path is a *π-extending Hamiltonian path* if its order is a linear extension of π. Using this definition we can generalize the classical Hamiltonian Path using precedence constraints.

▶ **Problem 1.** Partially Ordered Hamiltonian Path Problem (POHPP)
Instance: A graph G, a partial order π on the vertex set of G.
Question: Is there an ordered Hamiltonian path $(v_1, \ldots, v_n)$ in G such that for all $i, j \in [n]$ it holds that if $(v_i, v_j) \in \pi$, then $i \leq j$?

A similar definition can be derived for Hamiltonian Cycle.

▶ **Problem 2.** Partially Ordered Hamiltonian Cycle Problem (POHCP)
Instance: A graph G, a partial order π on the vertex set of G.
Question: Is there an ordered Hamiltonian path $(v_1, \ldots, v_n)$ in G such that v_1 and v_n are adjacent and for all $i, j \in [n]$ it holds that if $(v_i, v_j) \in \pi$, then $i \leq j$?

The problems Hamiltonian Path and Hamiltonian Cycle without precedence constraints are independent from each other, i.e., there exist graph classes where one of the problems is trivial while the other is NP-hard. It is easy to show that Hamiltonian Path is NP-complete on the graphs that are not Hamiltonian. In contrast, Hamiltonian Cycle is NP-complete on traceable graphs [36, Lemma 21.18]. Using the same arguments as in [5] for graph classes, we can show that for POHPP and POHCP the status of being in FPT or XP is equivalent for many graph width parameters.

To this end, let $\mathbb{G}$ be the family of all graphs. Let $\xi : \mathbb{G} \to \mathbb{N}$ be a graph width parameter. A *graph operation* Ω is a function $\mathbb{G} \to \mathbb{G}$. We say that ξ is *closed under graph operation* Ω if there is a computable function $h : \mathbb{N} \to \mathbb{N}$ such that for all graphs G, it holds that $\xi(\Omega(G)) \leq h(\xi(G))$.

▶ **Observation 2.1** (see Observation 2.4 [5]). *Let ξ be a graph width parameter that is closed under the addition of a universal vertex. Then there is a polynomial-time* FPT*-reduction from the POHPP parameterized by ξ to the POHCP parameterized by ξ.*

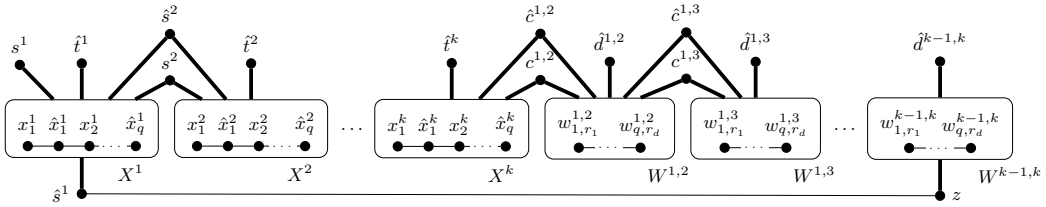


Figure 2 Construction of Theorem 3.1. If a vertex is adjacent to a box via a thick edge, then the vertex is adjacent to all vertices in that box. For *distance to path*, the ends of consecutive subpaths of the X^i and $W^{i,j}$ are adjacent, for *distance to linear forest modules* they are not adjacent.

For the converse direction, we are not able to give a (many-one) FPT-reduction. Nevertheless, we can give an FPT-Turing reduction. Furthermore, we do not need to restrict the parameter ξ in any form.

▶ **Observation 2.2** (see Observation 2.3 [5]). *Let ξ be a graph width parameter. If we can solve (Min)POHPP in time $f(\xi(G)) \cdot n^{g(\xi(G))}$ on a graph G, then we can solve (Min)POHCP in time $f(\xi(G)) \cdot n^{g(\xi(G))+2}$ on G.*

Note that Observation 2.1 does also hold for the classical problems Hamiltonian Path and Hamiltonian Cycle. In contrast, Observation 2.2 does not hold for the classical problems since Hamiltonian Cycle is NP-complete on traceable graphs [36, Lemma 21.18].

These two observations allow us to restrict our proofs to the path case. Whenever we prove W[1]-hardness for POHPP, the respective width parameter is closed under the addition of a universal vertex. Therefore, Observation 2.1 also implies W[1]-hardness for the cycle case. If we present some FPT or XP algorithm for (Min)POHPP, then Observation 2.2 also implies an FPT or XP algorithm for (Min)POHCP.

3 Sparse Width Parameters

We start with parameters that are unbounded for cliques. As we have shown in [5], both the POHPP and the POHCP are para-NP-hard when parameterized by *treewidth*. Therefore, we focus on parameters that form upper bounds of *treewidth*.

3.1 Hardness

We start by showing that even for very restricted *distance to $\mathcal{G}$* parameters, there is no hope for FPT algorithms.

▶ **Theorem 3.1.** *POHPP and POHCP are W[1]-hard when parameterized by either distance to path or distance to linear forest modules.*

Proof. We sketch the reduction for *distance to path* from the W[1]-hard Multicolored Clique Problem (MCP) [25, 44]. In MCP the input is a graph G whose vertex set is partitioned into k color classes $V_1, \dots, V_k$ with $V_i := \{v_1^i, \dots, v_q^i\}$ for all $i \in [k]$. MCP asks whether there is a clique in G containing exactly one vertex per color class. Given an instance of MCP, we construct a graph G' using the following gadgets (see Figure 2):

Selection Gadget For every color class $i \in [k]$, we have a selection gadget consisting of a vertex $\hat{t}^i$ and a path X^i that contains for every $v_p^i \in V_i$ a vertex x_p^i. These vertices are ordered by their index p and after every vertex x_p^i (and, thus, before x_{p+1}^i) there is a vertex $\hat{x}_p^i$ in that path. The vertex $\hat{t}^i$ is adjacent to all vertices in the path X^i.

Verification Gadget For each pair $i, j \in [k]$ with $i < j$, we have a verification gadget consisting of a vertex $\hat{d}^{i,j}$ and a path $W^{i,j}$ containing for every edge $v_p^i v_r^j \in E(G)$ a vertex $w_{p,r}^{i,j}$ in arbitrary order. The vertex $\hat{d}^{i,j}$ is adjacent to all vertices in the path $W^{i,j}$.

We order the subpaths described in the selection and verification gadgets as follows:

$$X^1, X^2, \ldots, X^k, W^{1,2}, W^{1,3}, \ldots, W^{1,k}, W^{2,3}, W^{2,4}, \ldots, W^{k-1,k}.$$

The last vertex of one of the paths is adjacent to the first vertex of the succeeding path and, thus, they form one large path Ψ in G'. Furthermore, G' contains vertices s^1 and $\hat{s}^1$ that are adjacent to all vertices in X^1. For every $i \in [k]$ with $i > 1$, G' contains vertices s^i and $\hat{s}^i$ that are adjacent to all vertices in X^i and X^{i-1}. For all $i, j \in [k]$ with $i < j$, the graph contains vertices $c^{i,j}$ and $\hat{c}^{i,j}$ that are adjacent to all vertices in $W^{i,j}$ and to all vertices in the path preceding $W^{i,j}$ in the ordering described above. Finally, G contains a vertex z that is adjacent to all vertices in $W^{k-1,k}$ and to $\hat{s}^1$. As there are $\mathcal{O}(k^2)$ vertices in G' that are not part of the path Ψ, the *distance to path* of G' is $\mathcal{O}(k^2)$.

We define Y as the set containing all the vertices defined above that have a hat in their name, i.e., $Y := \{\hat{s}^i \mid i \in [k]\} \cup \{\hat{t}^i \mid i \in [k-1]\} \cup \{\hat{c}^{i,j}, \hat{d}^{i,j} \mid i, j \in [k], i < j\}$. The partial order π is the reflexive and transitive closure of the following constraints:

(P1) $s^1 \prec v$ for all $v \in V(G') \setminus \{s^1\}$,

(P2) $z \prec y$ for all $y \in Y$,

(P3) $x_p^i \prec w_{p,r}^{i,j}$ for all $i, j \in [k]$ with $i < j$ and all $p, r \in [q]$ with $v_p^i v_r^j \in E(G)$

(P4) $x_r^j \prec w_{p,r}^{i,j}$ for all $i, j \in [k]$ with $i < j$ and all $p, r \in [q]$ with $v_p^i v_r^j \in E(G)$

Every π-extending Hamiltonian path has to visit exactly one vertex in each X^i and $W^{i,j}$ on its way from s^1 to z as all the vertices with hats cannot be visited before z. Due to the constrains (P3) and (P4), a vertex of $W^{i,j}$ corresponding to an edge $v_p^i v_r^j$ can only be visited if both x_p^i and x_r^j have already been visited. Therefore, we only can reach z if the visited vertices of the selection gadgets form a a multicolored clique of G.

Conversely, a multicolored clique implies that we can reach z from s^1 using such a path. We can visit the remaining vertices by traversing the path Ψ and using $\hat{c}^{i,j}, \hat{d}^{i,j}, \hat{s}^i$ and $\hat{t}^i$ outside of Ψ to jump over vertices that have already been visited. For details see the full version [6]. For *distance to linear forest modules* we can adapt the construction by deleting the edges between consecutive gadgets since these edges have not been used in the proof. The resulting graph has *distance to linear forest modules* $\mathcal{O}(k^2)$. ◀

3.2 Algorithms

We start this section with some easy observations about width values of traceable graphs. A traceable graph with *vertex cover number* k has at most $2k + 1$ vertices. Therefore, MinPOHPP and MinPOHCP parameterized by *vertex cover number* allow kernels with a linear number of vertices and, thus, FPT algorithms. It follows from Lemma 6.2 and Equation 6.2 by Nešetřil and de Mendez [41] that a traceable graph of *treedepth* k has less than 2^k vertices. As both MinPOHPP and MinPOHCP can be solved in exponential time [7], there are simple double-exponential FPT algorithms for *treedepth*. For the *feedback edge set number* we can give a single-exponential FPT algorithm. As was shown in [16], an n-vertex graph of *feedback edge set number* k has at most $2^k \cdot \binom{n}{2}$ different paths.[4] Enumerating these paths leads to the following result.

[4] Note that the result is only given in the arXiv version of the paper.

▶ **Theorem 3.2.** *MinPOHPP and MinPOHCP can be solved in $2^k \cdot n^{\mathcal{O}(1)}$ time on an n-vertex graph of feedback edge set number k.*

Unless $\mathsf{W}[1] = \mathsf{FPT}$, this result cannot be extended to *feedback vertex set number* since POHPP and POHCP are $\mathsf{W}[1]$-hard for this parameter, due to Theorem 3.1. However, we can give an XP algorithm for MinPOHPP when parameterized by *distance to outerplanar*, a more general parameter than *feedback vertex set number*. Before we present this algorithm, we first focus on a more restricted case that allows an FPT algorithm.

It has been shown by Beisegel et al. [7] that MinPOHPP can be solved in $\mathcal{O}(n^2)$ time on outerplanar graphs. The key ingredient of that algorithm is the observation that the vertices of the outer face contained in a prefix of a Hamiltonian path must be an *interval* of the outer face, i.e., a consecutive part of the closed walk that forms the face (see Lemma 5.2 in [7]). This observation does not only hold for outerplanar graphs but for all plane graphs and all faces. It is important to note that this does not mean that the prefix contains this interval as a subpath. In fact, there can be vertices in between that are not part of the boundary of the face. This general result allows us to extend the polynomial-time algorithm for outerplanar graphs to an FPT algorithm for plane graphs parameterized by the number of vertices that are not on the outer face. Note that our algorithm uses ideas similar to those of [17].

▶ **Theorem 3.3.** *Given a plane graph G with k vertices not on the outer face, we can solve MinPOHPP on G in $\mathcal{O}(2^k kn^3)$ time and MinPOHCP on G in $\mathcal{O}(2^k kn^5)$ time.*

Proof. We only sketch the idea of the algorithm as it is a quite straightforward adaption of Algorithm 2 in [7]. For details, we refer to that publication. First note that an induced subgraph does not have more inner vertices for a fixed embedding. Hence, it suffices to deal with 2-connected graphs (Theorem 5.1 in [7]). Let $C = (v_0, \dots, v_\ell)$ be the outer face of the planar embedding and let W be the set of vertices that are not on the outer face. Since G is 2-connected, the face C forms a cycle. Furthermore, let c be the cost function on the edges of G.

The idea of Algorithm 2 in [7] is to use a dynamic programming approach. To this end, the authors consider tuples (a, b, ω), which represent prefixes of Hamiltonian paths. Here, a and b are the indices of the endpoints of the interval on C associated with that prefix (see Lemma 5.2 in [7]) and ω is either 1 or 2 depending on whether the prefix ends in v_a or v_b. For the problem considered here, we simply need to adjust these to tuples of the form (a, b, X, t), where a and b again represent the indices of the endpoints of the interval on C associated with the prefix, $X \subseteq W$ is the set of inner vertices in the prefix and t forms the endpoint of the prefix. There are at most n^2 intervals and at most 2^k subsets of W. The endpoint t either has to be a vertex of X or one of the vertices v_a and v_b. Therefore, there are at most $(k+2)$ endpoints and the total number of tuples is bounded by $2^k(k+2)n^2$. Similar as in Algorithm 2 of [7], we compute for every tuple (a, b, X, t) the minimal cost $M(a, b, X, t)$ of an ordered path $\mathcal{P}$ of G fulfilling the following properties (or ∞ if no such path exists):

- **(i)** $\mathcal{P}$ consists of the vertices in the interval between v_a and v_b of C and in the set X,
- **(ii)** $\mathcal{P}$ is a prefix of a linear extension of π,
- **(iii)** $\mathcal{P}$ ends in t.

This is done inductively by the size of $\mathcal{P}$. Let a', b', and X' be the updated values if t is removed from the potential prefix. We first have to check whether t is minimal in π if all the vertices of $[a', b']$ and X' have been visited. This costs $\mathcal{O}(n)$ time. If this is not the case, we set the M-value to ∞. Otherwise, we check the M-values for all possible vertices t' that are before t in the prefix. Note that t' can only be an element of X' or one of the two

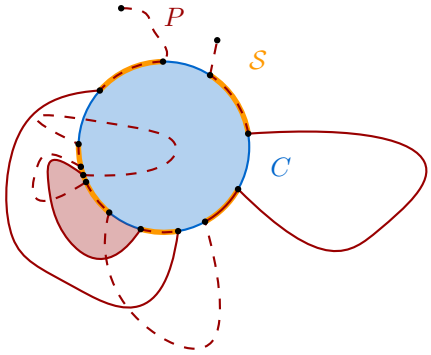


Figure 3 A face C (blue) and a prefix P of a Hamiltonian path (red). The solid parts of P are crests. The dashed parts might contain vertices of W and are thus not necessarily planar.

vertices $v_{a'}$ and $v_{b'}$, due to Lemma 5.2 in [7]. For each choice of t' we compute the value $M(a', b', X', t') + c(tt')$ and set the M-value of (a, b, X, t) to the minimum of these values. Note that this can be done in $\mathcal{O}(k) \subseteq \mathcal{O}(n)$ time using an adjacency matrix.

Summing up, for each of the $\mathcal{O}(2^k kn^2)$ tuples we need $\mathcal{O}(n)$ time which leads to the overall running time of $\mathcal{O}(2^k kn^3)$. The proof of the correctness of the algorithm follows along the lines of the proof of Theorem 5.3 in [7]. ◀

The idea of that algorithm can also be used to give an XP algorithm for *distance to outerplanar*. The crucial difference in the complexity is based on the fact that there is not only one interval of the outer face used by a prefix of a Hamiltonian path but at most $k + 1$. To prove this, we need the following definitions.

Let C be a closed walk of a plane graph G. A subgraph H lies *inside* of C if all vertices and edges of H are elements of C or are placed in a bounded region of $\mathbb{R}^2 \setminus C$. The subgraph H lies *strictly inside* of C if it lies inside of C and does not contain a vertex of C. A subgraph H lies *outside* of C if all vertices and edges of H are elements of C or are placed in the unbounded region of $\mathbb{R}^2 \setminus C$.

▶ **Lemma 3.4.** *Let G be a graph with a set $W \subseteq V(G)$ of size k such that $G - W$ is planar. Let C be a face of a planar embedding of $G - W$ and let $V(C)$ be the vertices on the boundary of C. Let P be a prefix of a Hamiltonian path of G. Then there exists a partition of $V(C) \cap V(P)$ into at most $k + 1$ intervals of C.*

Proof. We use induction over the number of vertices of G. For one vertex the statement is trivial. Assume that the statement holds for all graphs with less than n vertices. Let G be a graph with n vertices and a set $W \subseteq V(G)$ of size k such that $G - W$ is planar. Let C be a face of a planar embedding of $G - W$. Note that C could be a closed walk with repeating vertices, for example, the outer face of a graph that is not 2-connected. To simplify the notation, we assume that C does not repeat vertices. By tweaking the definitions, the other case can be proven analogously.

Let P be a prefix of a Hamiltonian path $\mathcal{P}$ of G that does not contain all vertices of $\mathcal{P}$. Let $\mathcal{S}$ be the smallest partition of $V(C) \cap V(P)$ into intervals of C, i.e., the intervals are inclusion-maximal. See Figure 3 for an illustration. Let $|\mathcal{S}| = \ell$. If we remove the elements of C from P, we get pairwise disjoint subpaths of P and the endpoints of these subpaths have neighbors on P that lie on C. We ignore the subpath that contains all vertices before the first vertex of C on P if it exists and the subpath containing all vertices after the last vertex of C on P if it exists. We also ignore all subpaths where the neighbor of the first and the

last vertex lie on the same interval of $\mathcal{S}$. Thus, we only consider subpaths that connect two intervals of $\mathcal{S}$. As all intervals of $\mathcal{S}$ have to be connected via such paths, there are at least $\ell - 1$ of these subpaths. If each of these subpaths contains a vertex of W, then $\ell \leq k+1$ and we are done. Otherwise, consider a subpath P_{sub} that does not contain vertices of W. We define the *crest* induced by P_{sub} as the subpath of P containing P_{sub} and the P-neighbors of its endpoints. Let Q be such a crest. Let v_i be the first vertex and v_j be the last vertex of Q with regard to the ordering of P. By definition, both v_i and v_j lie on C. We define $C[v_i, v_j]$ as the part of C between v_i and v_j in the cyclic ordering and $C[v_j, v_i]$ as the part of C between v_j and v_i in the cyclic ordering. Further note that v_i and v_j lie in different intervals of $\mathcal{S}$. Therefore, there are vertices not in P, both in $C[v_i, v_j]$ and in $C[v_j, v_i]$. Let t be the end vertex of P or, in case this end vertex is v_j, let t be the successor of v_j in the Hamiltonian path $\mathcal{P}$. Note that t exists as P does not contain all vertices of $\mathcal{P}$. W.l.o.g. we may assume that t lies outside of $\widetilde{Q} = Q \cup C[v_i, v_j]$. If this is not the case, then we define one face inside of $\widetilde{Q}$ as outer face without changing the faces of the embedding. Then, we can consider $Q \cup C[v_j, v_i]$ as $\widetilde{Q}$.

Let R be a *minimal* crest inside of $\widetilde{Q}$, i.e., there is no crest that lies inside of $\widetilde{R} = R \cup C[x, y]$, where x and y are the endpoints of R and $C[x, y]$ is the part of C between x and y that is inside of $\widetilde{Q}$. Note that $C[x, y]$ contains vertices that are not part of P as the crest R connects two intervals, say S_x and S_y where $x \in S_x$ and $y \in S_y$.

We will now show that we can always find either a vertex to delete or an edge to contract in order to apply the induction hypothesis.

Assume that there is an edge $ab \in E(P)$ such that w.l.o.g. b is strictly inside of $\widetilde{R}$. We contract the edge ab in G and P. The resulting graph G' and resulting path P' fulfill the induction hypothesis, i.e., $W \subseteq V(G')$, P' is a prefix of a Hamiltonian path of G' and C is a face of an embedding of G'. Therefore, we can partition $V(C) \cap V(P')$ into at most $k+1$ intervals of C. These intervals map bijectively to intervals of $V(C) \cap V(P)$. Hence, we can assume that P does not contain any vertices strictly inside of $\widetilde{R}$.

Suppose there is an interval $S \in \mathcal{S}$ that is contained completely in $C[x, y] - \{x, y\}$. As $\widetilde{R}$ separates S from the vertices outside of $\widetilde{R}$ and no vertices in P are strictly inside of $\widetilde{R}$, all P-neighbors of vertices of S are either in S or in W. As t lies outside of $\widetilde{R}$, the interval S has at least one P-neighbor in W. If S has exactly one P-neighbor $w \in W$ – i.e. P starts with S – then we delete w and the whole interval S and connect the neighbors of S on C with an edge. This gives a graph G' with one less vertex in W and one less interval. The suffix of P that starts at the successor of w is a prefix of a Hamiltonian path in G'. The statement follows by induction.

Suppose S has more than one P-neighbor in W. We connect these neighbors to form a clique. Furthermore, we sort them by their appearance in P and contract the last and the penultimate vertex of these neighbors. As t lies outside of $\widetilde{R}$, the last neighbor of S cannot be used to enter $\tilde{R}$. Therefore, all vertices visited in P between these two neighbors lie in S. We now delete S and connect its neighbors on C with an edge. We can transform P into P' by replacing a sequence of S-vertices between two P-neighbors of S by one of the clique edges. The statement follows by induction as there is one vertex less in W and one interval less, see Figure 4.

If there is no interval that is contained completely in $C[x, y] - \{x, y\}$, then all vertices on $C[x, y]$ are either in S_x, S_y or not in P. In this case, we can apply similar arguments to contract the vertices that are not in P and, thus, join S_x and S_y to reduce the number of intervals. The details can be found in the full version [6]. ◀

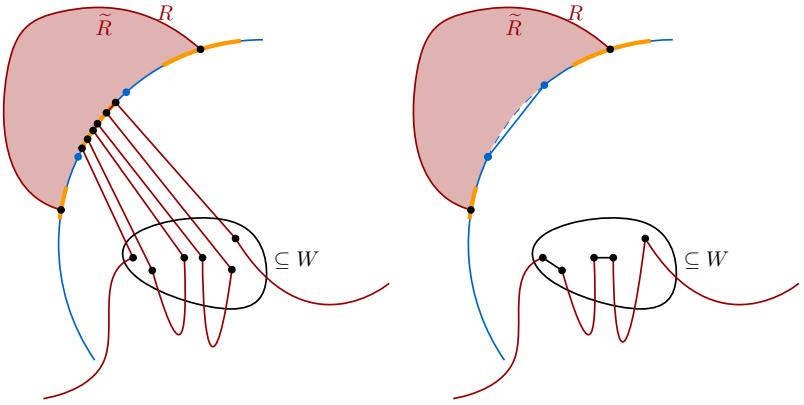


Figure 4 Illustration of Lemma 3.4.

Using this lemma, we can adapt the algorithm given in Theorem 3.3 such that it solves the MinPOHPP in XP time for *distance to outerplanar*. This adaption is rather straightforward. In the dynamic program, we replace the vertices representing the single interval on C by a set of at most $k+1$ intervals. It is not difficult to observe that the number of those sets is bounded by $n^{\mathcal{O}(k)}$. This leads to the following running time bound.

▶ **Theorem 3.5.** *Given an n-vertex graph G with distance to outerplanar k, we can solve MinPOHPP and MinPOHCP on G in $n^{\mathcal{O}(k)}$ time.*

4 Non-Sparse Width Parameters

All parameters considered so far are sparse, i.e. unbounded for cliques. As POHPP and POHCP are trivial on cliques, it makes sense to consider also non-sparse parameters. Both MinPOHPP and MinPOHCP are NP-hard on cliques as they form generalizations of (Path) TSP. Thus, we only consider the unweighted problems in this section.

4.1 Hardness

As mentioned in the introduction, Hamiltonian Path and Hamiltonian Cycle can be solved in FPT time when parameterized by the *independence number*, that is the size of the largest independent set [26]. For traceable graphs with at least two vertices, this parameter is a lower bound on the *edge clique cover number*.[5] Thus, both problems can also be solved in FPT time for this parameter. Here, we show that these results cannot be extended to POHPP or POHCP unless P = NP.

▶ **Theorem 4.1.** *POHPP and POHCP are NP-complete on graphs of edge clique cover number 3.*

Proof. We reduce from a special case of the Alternating Linear Extension Problem that was introduced and shown to be NP-complete in [7]. In this case of the problem, we are given two non-empty disjoint sets A and B with $|A| = |B|$ and a partial order $\pi \subseteq A \times B$ and we ask for a linear extension of π that alternates between elements of A and B starting in some vertex of A.

[5] This does not only hold for traceable graphs but for all graphs that do not contain isolated vertices. An edge clique cover of such a graph is also a clique cover of the graph and an independent set can contain at most one vertex of every clique in that clique cover.

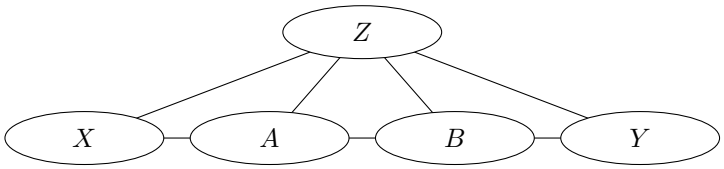


Figure 5 Construction of Theorem 4.1. The ellipses form cliques. The connection of an ellipse with another implies that all edges between the vertices of the two cliques are present.

Let (A, B, π) be an instance of this problem with $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_n\}$. We build a graph G as follows (see Figure 5 for an illustration). The vertex set of G is the disjoint union of A, B, $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, y_n\}$, and $Z = \{z_1, \ldots, z_n\}$. The graph G contains all edges except for those between X and B, between X and Y as well as those between Y and A. In other words, the edges of G can be covered by the three cliques $C_1 = A \cup X \cup Z$, $C_2 = A \cup B \cup Z$, and $C_3 = B \cup Y \cup Z$.

The partial order π' contains all tuples of π. Additionally, it fixes x_1 as start vertex and contains the following chain of constraints: $x_1 \prec y_1 \prec z_1 \prec x_2 \prec y_2 \prec z_2 \cdots \prec x_n \prec y_n \prec z_n$.

Assume the instance (A, B, π) has an alternating linear extension σ. W.l.o.g. we may assume that $\sigma = (a_1, b_1, a_2, b_2, \ldots, a_n, b_n)$. Inserting x_i before a_i and (y_i, z_i) after b_i leads to a π'-extending Hamiltonian path and cycle of G.

Conversely, assume we have a π'-extending Hamiltonian path $\mathcal{P}$ in G. We observe that $\mathcal{P}$ uses one vertex of A and one vertex of B between x_i and y_i as all vertices z_j with $j < i$ are to the left of x_i and all vertices z_j with $j \geq i$ are to the right of y_i. Since X and Y have the same size as A and B, we have to use exactly one vertex of A and one of B between every x_i and y_i. Thus, $\mathcal{P}$ contains an alternating linear extension of π. ◀

We have to leave open the case of *edge clique cover number* 2. However, we can observe that the graph G constructed in the proof of Theorem 4.1 has *clique cover number* 2 since the cliques C_1 and C_3 cover all vertices. This implies the following result.

▶ **Corollary 4.2.** *POHPP and POHCP are* NP*-complete on graphs of clique cover number 2.*

Note that for graphs of *clique cover number* 1 (i.e., cliques), POHPP and POHCP are trivial as every linear extension of π induces a Hamiltonian cycle.

It follows from Theorem 3.1 that POHPP and POHCP are W[1]-hard when parameterized by *distance to block*. Here, we extend this result to *distance to clique*.

▶ **Theorem 4.3.** *POHPP and POHCP are* W[1]*-hard when parameterized by either distance to clique or distance to cluster modules also known as twin-cover number.*

Proof. As for Theorem 3.1, we use a reduction from Multicolored Clique to POHPP and then use Observation 2.1 to derive W[1]-hardness also for POHCP. Let G be an instance of MCP with color classes $V_1, \ldots, V_k$ where $V_i = \{v_1^i, \ldots, v_q^i\}$. The construction works similar as for Theorem 3.1. The main difference is how we encode the selection of a vertex from a color class. While in the proof of Theorem 3.1 the representative of the selected vertex was visited, here we visit all representatives except from the one of the selected vertex.

We construct a graph G' using the following gadgets:

Selection Gadget For every color class $i \in [k]$, we have a clique X^i that contains for every $v_p^i \in V_i$ a vertex x_p^i.

Verification Gadget For each pair $i, j \in [k]$ with $i < j$, we have a clique $W^{i,j}$ containing for every edge $v_p^i v_r^j \in E(G)$ a vertex $w_{p,r}^{i,j}$.

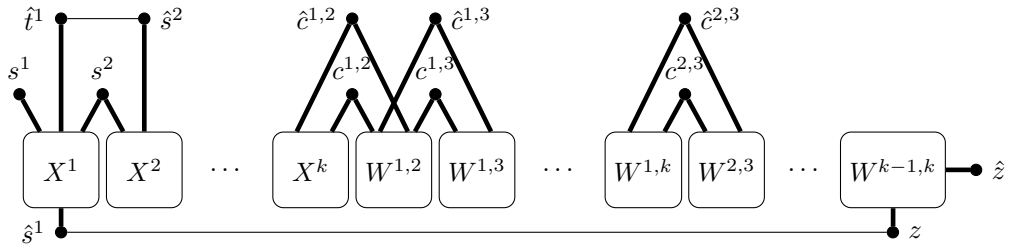


Figure 6 Construction of Theorem 4.3. If a vertex is adjacent to a box via a thick edge, then the vertex is adjacent to all vertices in that box. For *distance to clique*, all the boxes are pairwise adjacent, for *distance to linear forest modules* they are not adjacent.

Next we describe how these gadgets are connected to each other (see Figure 6). The union of the selection gadgets and the verification gadgets forms one large clique. We order the subcliques described in the selection and verification gadgets as follows:

$$X^1, X^2, \ldots, X^k, W^{1,2}, W^{1,3}, \ldots, W^{1,k}, W^{2,3}, W^{2,4}, \ldots, W^{k-1,k}.$$

We have the following additional vertices in G'. First, we have s^1, $\hat{s}^1$, and $\hat{t}^1$ that are adjacent to all vertices in X^1. For every i with $2 \leq i \leq k-1$, we have vertices s^i, $\hat{s}^i$ and $\hat{t}^i$. For $i = k$, we have only s^i and $\hat{s}^i$. Vertex s^i is adjacent to all vertices in X^i and X^{i-1} and the vertices $\hat{s}^i$ and $\hat{t}^i$ are adjacent to all vertices in X^i. Furthermore, $\hat{s}^i$ is adjacent to $\hat{t}^{i-1}$. For all $i, j \in [k]$ with $i < j$, there are vertices $c^{i,j}$ and $\hat{c}^{i,j}$ that are adjacent to all vertices in $W^{i,j}$ and to all vertices in the clique to left of $W^{i,j}$ in the ordering described above. Finally, we have vertices z and $\hat{z}$ that are adjacent to all vertices in $W^{k-1,k}$. Furthermore, z is adjacent to $\hat{s}^1$. Observe that the graph G' without the selection and verification gadgets contains $3k + 2\binom{k}{2} + 2$ vertices. Therefore, the *distance to clique* of G' is $\mathcal{O}(k^2)$.

The partial order π is the reflexive and transitive closure of the following constraints.

(P1) $s^1 \prec v$ for all $v \in V(G') \setminus \{s^1\}$,
(P2) $v \prec \hat{z}$ for all $v \in V(G') \setminus \{\hat{z}\}$,
(P3) $s^1 \prec s^2 \prec \cdots \prec s^k$,
(P4) $z \prec \hat{s}^1 \prec \hat{t}^1 \prec \hat{s}^2 \prec \hat{t}^2 \prec \cdots \prec \hat{s}^k$,
(P5) $c^{i,j} \prec c^{i',j'}$ for all $i, j, i', j' \in [k]$ with $i < i'$ or $i = i'$ and $j < j'$,
(P6) $c^{k-1,k} \prec z$,
(P7) $c^{i,j} \prec w^{i,j}_{p,r}$ for all $i, j \in [k]$ with $i < j$ and all $w^{i,j}_{p,r} \in W^{i,j}$,
(P8) $x^i_p \prec w^{i,j}_{r,s}$ for all $i, j \in [k]$ with $i < j$ and all $w^{i,j}_{r,s} \in W^{i,j}$ with $p \neq r$,
(P9) $x^j_p \prec w^{i,j}_{r,s}$ for all $i, j \in [k]$ with $i < j$ and all $w^{i,j}_{r,s} \in W^{i,j}$ with $p \neq s$.

First assume that there is a multicolored clique $\{v^1_{p_1}, \ldots, v^k_{p_k}\}$ in G. Then we claim that there is π-extending Hamiltonian path in G'. We start our path in s^1 and then we visit all the vertices in X^1 except for the vertex $x^1_{p_1}$. Then we visit s^2. We repeat this for all $i \in [k]$. Next we visit $c^{1,2}$. Now we visit $w^{1,2}_{p_1,p_2}$. Note that this is possible since all the vertices that have to be to the left of $w^{1,2}_{p_1,p_2}$ by (P8) and (P9) are already visited. Next we visit $c^{1,3}$ and afterwards $w^{1,3}_{p_1,p_3}$ which is possible for the same reason as mentioned above. We repeat this procedure until we reach $w^{1,k}_{p_1,p_k}$. Next, we visit $c^{2,3}$. The same procedure works until we reach $w^{k-1,k}_{p_{k-1},p_k}$. Next we visit z. Now, starting with $i = 1$, we visit for all $i \in [k]$ the vertices $\hat{s}^i$, followed by $x^i_{p_i}$ and $\hat{t}^i$ until we reach $x^k_{p_k}$. Then, we visit $\hat{c}^{1,2}$ followed by all remaining vertices $w^{1,2}_{p,r} \in W^{1,2}$ which is possible as all the left vertices in the constraints (P8) and (P9)

have already been visited. We repeat this for all i, j with $i < j$ in the lexicographic order and eventually we have visited all vertices in $W^{k-1,k}$. Finally, we visit $\hat{z}$. It is easy to check that this Hamiltonian paths fulfills all constraints of π.

Now assume that there is a π-extending Hamiltonian path $\mathcal{P}$ in G'. Let $i \in [k]$ be arbitrary. By (P4), $\hat{s}^i$ is to the right of z. By (P2), $\hat{s}^i$ is not the last vertex of $\mathcal{P}$ and, thus, it has two neighbors in $\mathcal{P}$. One of these two neighbors has to be in X^i. Therefore, for each $i \in [k]$, there is at least one vertex in X^i that is to the right of z in $\mathcal{P}$.

Using the constraints (P8) and (P9), we can observe that there can be at most one vertex of $W^{i,j}$ to the left of z for every choice of i and j with $i < j$. Otherwise, all vertices of X^i and X^j would be to the left of z contradicting our observation above. This also implies that there is exactly one vertex of $W^{i,j}$ to the left of z since $c^{i,j}$ has two neighbors in $\mathcal{P}$ and one of these neighbors has to be an element of $W^{i,j}$. The constraints (P8) and (P9) imply now that there can be at most one vertex of every X^i to the right of z. Combining this with the existence of such a vertex shown above, we see that in every X^i there is exactly one vertex $x^i_{p_i}$ to the right of z. Since there is a vertex of every $W^{i,j}$ to the left of z, it is not difficult to see that the set $\{v^1_{p_1}, \dots, v^k_{p_k}\}$ forms a multicolored clique in G.

The arguments above show that the graph (G', π) is a valid reduction. Since the *distance to clique* of G' is $\mathcal{O}(k^2)$ it also a proper FPT reduction from MCP to POHPP parameterized by the *distance to clique*.

Similar as in Theorem 3.1, we can adapt the construction to show that POHPP is also W[1]-hard for *distance to cluster modules*. To this end, we delete all the edges in G' between different gadgets. This works since these edges have not been used in the proof and since the resulting graph has *distance to cluster modules* $\mathcal{O}(k^2)$. ◀

4.2 Algorithms

We start with an FPT algorithm for *edge distance to block* when given a respective edge set.

▶ **Theorem 4.4.** *POHPP and POHCP can be solved in* $(k+1)! \cdot 2^k \cdot n^{\mathcal{O}(1)}$ *time on an* n*-vertex graph when given a set* $F \subseteq E(G)$ *with* $|F| \leq k$ *and* $G - F$ *is a block graph.*

Proof. We consider all possible options for the start vertex of the Hamiltonian path and how the path traverses through the edges of F, i.e., we fix the set of edges $\widehat{F} \subseteq F$ that are used and also fix in which direction and in which order they are used. There are at most $k \cdot k! \cdot 2^k \cdot n$ many of these options. Let $\widehat{W}$ be the set of vertices incident to edges of $\widehat{F}$. Note that we only continue if every vertex in $\widehat{W}$ has at most two incident edges in $\widehat{F}$. Furthermore, the ordering on $\widehat{F}$ implies a vertex ordering σ on $\widehat{W}$.

First, we check whether σ is valid for π. If so, in the rest of the procedure for that option, we try to fill in all the other vertices of G. Whenever there are two consecutive vertices x and y in σ that are not adjacent via an edge of $\widehat{F}$, we have to use edges of $G - F$. Since the path between the blocks of these vertices in the block-cut tree of $G - F$ is unique, we know which cut vertices have to be used on that path and we reserve these cut vertices for that path. The algorithm now traverses the ordering σ and, if necessary, uses paths in $G - F$. These paths visit all vertices of a block that are possible due to π except those that are reserved for later paths. The details can be found in the full version [6]. ◀

Note that Dumas et al. [20] present a quadratic kernel for deciding whether the *edge distance to block* of a graph G is at most k. This implies an FPT algorithm with running time $k^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ that decides whether the *edge distance to block* is at most k. Calling this algorithm $\mathcal{O}(m)$ times, we can also construct the respective set F. This implies the following.

▶ **Corollary 4.5.** *POHPP and POHCP can be solved in* $((k+1)! \cdot 2^k + k^{\mathcal{O}(k)}) \cdot n^{\mathcal{O}(1)}$ *time on an* n*-vertex graph with edge distance to block* k*.*

Next we give an XP algorithm for the *(vertex) distance to block* k*.* We can compute a set W of size $\leq k$ such that $G - W$ is a block graph in $n^{\mathcal{O}(k)}$ time. For each vertex of W, the algorithm fixes its predecessor and successor in a potential Hamiltonian path. This choice can be transferred to an instance of POHPP with *edge distance to block* $\leq k$ with a fixed edge choice $\widehat{F}$. Then, it is sufficient to apply the subroutine of Theorem 4.4.

▶ **Theorem 4.6.** *If the distance to block of a graph* G *with* n *vertices is* k*, then POHPP and POHCP can be solved in* $n^{\mathcal{O}(k)}$ *time.*

As we have seen in Theorem 4.3, there is no FPT algorithm for *distance to cluster modules* unless W[1] = FPT. However, we can give such an algorithm for *distance to clique module*.

▶ **Theorem 4.7.** *Given an* n*-vertex graph* G *with distance to clique module* k*, POHPP and POHCP can be solved in time* $k! \cdot n^{\mathcal{O}(1)}$*.*

Proof. Let $W \subseteq V(G)$ be a set of size k such that $G - W$ is a clique module of G. Note that such a set can be computed in polynomial time since the largest clique module is equivalent to the largest set of vertices with the same closed neighborhood. We now consider every ordering $\sigma = (w_1, \ldots, w_k)$ of the vertices in W that fulfills the constraints of π. To decide whether σ is an eligible choice, we have to check whether we can insert the vertices of $G - W$ into σ in such a way that the result is a π-extending Hamiltonian path. Some of the vertices of W may be *secluded*, i.e., they are not adjacent to any vertices in $G - W$. As secluded vertices have to induce subpaths in σ, we can contract them and only consider non-secluded vertices of W in the following. By adapting the partial order, we prevent that some vertex of $G - W$ is added between the vertices of the edge that results from that contraction.

We try to insert vertices of $G - W$ into the ordering σ as rightmost as possible, i.e., we try to only insert such a vertex into the gap between w_{i-1} and w_i if it is a predecessor of w_i in π. However, it might be that w_{i-1} and w_i are not adjacent and all predecessors of w_i in $G - W$ are already visited. Then we have to take some other vertex from $G - W$ between w_{i-1} and w_i. For all $x \in V(G - W)$, we define $r(x)$ as the minimum index j such that x is to the left of w_j in the partial order. Among all remaining vertices of $G - W$ that are allowed to be to the left of w_i, we take a vertex with minimal $r(x)$ value. The details of the algorithm and the proof of correctness can be found in the full version [6]. ◀

5 Conclusion

We have presented a comprehensive classification of the parameterized complexity of finding Hamiltonian paths or cycles with precedence constraints from the perspective of graph width parameters (see Figure 1). We were able to prove that all our XP algorithms cannot be extended to FPT algorithms (unless W[1] = FPT). Nevertheless, our W[1]-hardness proofs cannot be used to show – under the assumption of the Exponential Time Hypothesis – that the algorithms' exponents are asymptotically optimal since these proofs increase the parameter value quadratically. Therefore, one may ask whether there are also FPT-reductions where the parameter increases only linearly.

So far, our work has focused on the existence of parameterized algorithms. As a next step, one could also consider the question whether the FPT results could be extended to kernels of polynomial size. As mentioned, there is a trivial polynomial kernel for *vertex cover number*. We are rather optimistic that such a kernel can also be given for *feedback edge set number*.

One could also try to generalize FPT or XP algorithms to distance parameters for more general graph classes. One option are the *series-parallel graphs*, a generalization of outerplanar graphs. These graphs are exactly the graphs of *treewidth* 2. Note that a generalization to *distance to treewidth 3* is hopeless as POHPP and POHCP are already para-NP-hard for *edge distance to bandwidth 3* [5]. Another option would be to consider graph width parameters not on all graphs but only on some subclass. In particular, the planar case of *distance to outerplanar* could be interesting as it lies between the general W[1]-hard case and the FPT-case where the deleted vertices all lie in the interior of the outer face.

As POHPP and POHCP are already hard for very restricted graphs, one may ask how their complexity behaves if we restrict both the graph and the partial order. It has been shown [7] that POHPP is in XP and W[1]-hard when parameterized by the partial order's *width*. Using similar arguments as for Observations 2.1 and 2.2, we can show that the same holds for POHCP. However, it is open what happens if the problems are parameterized by both the partial order's *width* and some graph width parameter such as *treewidth* or *distance to clique*.

References

1 Zakir Hussain Ahmed and S. N. Narahari Pandit. The travelling salesman problem with precedence constraints. *Opsearch*, 38(3):299–318, 2001. doi:10.1007/BF03398638.

2 Norbert Ascheuer, Laureano F. Escudero, Martin Grötschel, and Mechthild Stoer. A cutting plane approach to the sequential ordering problem (with applications to job scheduling in manufacturing). *SIAM Journal on Optimization*, 3(1):25–42, 1993. doi:10.1137/0803002.

3 Katerina Asdre and Stavros D. Nikolopoulos. The 1-fixed-endpoint path cover problem is polynomial on interval graphs. *Algorithmica*, 58(3):679–710, 2010. doi:10.1007/s00453-009-9292-5.

4 Katerina Asdre and Stavros D. Nikolopoulos. A polynomial solution to the k-fixed-endpoint path cover problem on proper interval graphs. *Theoretical Computer Science*, 411(7):967–975, 2010. doi:10.1016/j.tcs.2009.11.003.

5 Jesse Beisegel, Katharina Klost, Kristin Knorr, Fabienne Ratajczak, and Robert Scheffler. A graph width perspective on partially ordered Hamiltonian paths and cycles I: Treewidth, pathwidth, and grid graphs, 2025. arXiv:2506.23790.

6 Jesse Beisegel, Katharina Klost, Kristin Knorr, Fabienne Ratajczak, and Robert Scheffler. A graph width perspective on partially ordered Hamiltonian paths and cycles II: Vertex and edge deletion numbers, 2025. arXiv:2510.08378.

7 Jesse Beisegel, Fabienne Ratajczak, and Robert Scheffler. Computing Hamiltonian paths with partial order restrictions. *ACM Transactions on Computation Theory*, 17(1), 2025. doi:10.1145/3711844.

8 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.

9 Lucio Bianco, Aristide Mingozzi, Salvatore Ricciardelli, and Massimo Spadoni. Exact and heuristic procedures for the traveling salesman problem with precedence constraints, based on dynamic programming. *INFOR: Information Systems and Operational Research*, 32(1):19–32, 1994. doi:10.1080/03155986.1994.11732235.

10 Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Information and Computation*, 243:86–111, 2015. doi:10.1016/J.IC.2014.12.008.

11 Charles J. Colbourn and William R. Pulleyblank. Minimizing setups in ordered sets of fixed width. *Order*, 1:225–229, 1985. doi:10.1007/BF00383598.

12 Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

13 Bruno Courcelle. The monadic second-order logic of graphs III: Tree-decompositions, minors and complexity issues. *Informatique Théorique et Applications/Theoretical Informatics and Applications*, 26(3):257–286, 1992. doi:10.1051/ITA/1992260302571.

14 Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Fast Hamiltonicity checking via bases of perfect matchings. *Journal of the ACM*, 65(3):12:1–12:46, 2018. doi:10.1145/3148227.

15 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michał Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. *ACM Transactions on Algorithms*, 18(2):17:1–17:31, 2022. doi:10.1145/3506707.

16 Erik D. Demaine, David Eppstein, Adam Hesterberg, Kshitij Jain, Anna Lubiw, Ryuhei Uehara, and Yushi Uno. Reconfiguring undirected paths. In Zachary Friggstad, Jörg-Rüdiger Sack, and Mohammad R. Salavatipour, editors, *Algorithms and Data Structures – 16th International Symposium, WADS 2019*, volume 11646 of *LNCS*, pages 353–365. Springer, 2019. arXiv:1905.00518, doi:10.1007/978-3-030-24766-9_26.

17 Vladimir G. Deĭneko, Michael Hoffmann, Yoshio Okamoto, and Gerhard J. Woeginger. The traveling salesman problem with few inner points. *Operations Research Letters*, 34(1):106–110, 2006. doi:10.1016/j.orl.2005.01.002.

18 Reinhard Diestel. *Graph Theory*, volume 173 of *GTM*. Springer, fourth edition, 2012. doi:10.1007/978-3-662-70107-2.

19 Martin Doucha and Jan Kratochvíl. Cluster vertex deletion: A parameterization between vertex cover and clique-width. In Branislav Rovan, Vladimiro Sassone, and Peter Widmayer, editors, *Mathematical Foundations of Computer Science 2012 – 37th International Symposium, MFCS 2012*, volume 7464 of *LNCS*, pages 348–359. Springer, 2012. doi:10.1007/978-3-642-32589-2_32.

20 Maël Dumas, Anthony Perez, Mathis Rocton, and Ioan Todinca. Polynomial kernels for edge modification problems towards block and strictly chordal graphs. *Discrete Mathematics & Theoretical Computer Science*, 27(2), 2025. doi:10.46298/DMTCS.12998.

21 Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite Model Theory*. Perspectives in Mathematical Logic. Springer, 1995. doi:10.1007/978-3-662-03182-7.

22 Laureano F. Escudero. An inexact algorithm for the sequential ordering problem. *European Journal of Operational Research*, 37(2):236–249, 1988. doi:10.1016/0377-2217(88)90333-5.

23 Laureano F. Escudero. On improving a solution to the ATSP with fixed origin and precedence relationships. *Trabajos de Investigacion Operativa*, 3:117–140, 1988. doi:10.1007/BF02888669.

24 Wolfgang Espelage, Frank Gurski, and Egon Wanke. How to solve NP-hard graph problems on clique-width bounded graphs in polynomial time. In Andreas Brandstädt and Van Bang Le, editors, *Graph-Theoretic Concepts in Computer Science, 27th International Workshop, WG 2001*, volume 2204 of *LNCS*, pages 117–128. Springer, 2001. doi:10.1007/3-540-45477-2_12.

25 Michael R. Fellows, Danny Hermelin, Frances Rosamond, and Stéphane Vialette. On the parameterized complexity of multiple-interval graph problems. *Theoretical Computer Science*, 410(1):53–61, 2009. doi:10.1016/j.tcs.2008.09.065.

26 Fedor V. Fomin, Petr A. Golovach, Nikola Jedličková, Jan Kratochvíl, Danil Sagunov, and Kirill Simonov. Path cover, Hamiltonicity, and independence number: An FPT perspective, 2025. arXiv:2403.05943.

27 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Intractability of clique-width parameterizations. *SIAM Journal on Computing*, 39(5):1941–1956, 2010. doi:10.1137/080742270.

28 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 Transactions on Algorithms*, 15(1), 2018. doi:10.1145/3280824.

29 Jakub Gajarský, Michael Lampis, and Sebastian Ordyniak. Parameterized algorithms for modular-width. In Gregory Z. Gutin and Stefan Szeider, editors, *Parameterized and Exact Computation – 8th International Symposium, IPEC 2013*, volume 8246 of *LNCS*, pages 163–176. Springer, 2013. doi:10.1007/978-3-319-03898-8_15.

30 Robert Ganian. Improving vertex cover as a graph parameter. *Discrete Mathematics & Theoretical Computer Science*, 17(2):77–100, 2015. doi:10.46298/dmtcs.2136.

31 Paul C. Gilmore, Eugene L. Lawler, and David B. Shmoys. Well-solved special cases. In Eugene L. Lawler, Jan Karel Lenstra, Alexander H. G. Rinnooy Kan, and David B. Shmoys, editors, *The traveling salesman problem. A guided tour of combinatorial optimization*, pages 87–143. Wiley, Chichester, 1985.

32 Petr A. Golovach, R. Krithika, Abhishek Sahu, Saket Saurabh, and Meirav Zehavi. Graph Hamiltonicity parameterized by proper interval deletion set. In Yoshiharu Kohayakawa and Flávio Keidi Miyazawa, editors, *LATIN 2020: Theoretical Informatics – 14th Latin American Symposium*, volume 12118 of *LNCS*, pages 104–115. Springer, 2020. doi:10.1007/978-3-030-61792-9_9.

33 Alon Itai, Christos H. Papadimitriou, and Jayme Luiz Szwarcfiter. Hamilton paths in grid graphs. *SIAM Journal on Computing*, 11(4):676–686, 1982. doi:10.1137/0211056.

34 Bart M. P. Jansen. *The Power of Data Reduction. Kernels for Fundamental Graph Problems*. PhD thesis, Utrecht University, 2013. URL: https://hdl.handle.net/1874/276438.

35 Bart M. P. Jansen, László Kozma, and Jesper Nederlof. Hamiltonicity below Dirac's condition. In Ignasi Sau and Dimitrios M. Thilikos, editors, *Graph-Theoretic Concepts in Computer Science – 45th International Workshop, WG 2019*, volume 11789 of *LNCS*, pages 27–39. Springer, 2019. doi:10.1007/978-3-030-30786-8_3.

36 Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer, 6th edition, 2018. doi:10.1007/978-3-662-56039-6.

37 Michael Lampis. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica*, 64(1):19–37, 2012. doi:10.1007/S00453-011-9554-X.

38 Peng Li and Yaokun Wu. A linear time algorithm for the 1-fixed-endpoint path cover problem on interval graphs. *SIAM Journal on Discrete Mathematics*, 31(1):210–239, 2017. doi:10.1137/140981265.

39 George B. Mertzios and Walter Unger. An optimal algorithm for the k-fixed-endpoint path cover on proper interval graphs. *Mathematics in Computer Science*, 3(1):85–96, 2010. doi:10.1007/s11786-009-0004-y.

40 Burkhard Monien and Ivan Hal Sudborough. Bounding the bandwidth of NP-complete problems. In Hartmut Noltemeier, editor, *Graphtheoretic Concepts in Computer Science – International Workshop, WG 80*, volume 100 of *LNCS*, pages 279–292. Springer, 1980. doi:10.1007/3-540-10291-4_20.

41 Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity. Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and Combinatorics*. Springer, 2012. doi:10.1007/978-3-642-27875-4.

42 Sophie N. Parragh, Karl F. Doerner, and Richard F. Hartl. A survey on pickup and delivery problems. Part I: Transportation between customers and depot. *Journal für Betriebswirtschaft*, 58(1):21–51, 2008. doi:10.1007/s11301-008-0033-7.

43 Sophie N. Parragh, Karl F. Doerner, and Richard F. Hartl. A survey on pickup and delivery problems. Part II: Transportation between pickup and delivery locations. *Journal für Betriebswirtschaft*, 58(2):81–117, 2008. doi:10.1007/s11301-008-0036-4.

44 Krzysztof Pietrzak. On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems. *Journal of Computer and System Sciences*, 67(4):757–771, 2003. doi:10.1016/S0022-0000(03)00078-3.

45 Harilaos N. Psaraftis. A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. *Transportation Science*, 14(2):130–154, 1980. doi:10.1287/trsc.14.2.130.

46 Sigve Hortemo Sæther and Jan Arne Telle. Between treewidth and clique-width. *Algorithmica*, 75(1):218–253, 2016. doi:10.1007/S00453-015-0033-7.

47 Sigve Hortemo Sæther. Solving Hamiltonian Cycle by an EPT algorithm for a non-sparse parameter. *Discrete Applied Mathematics*, 228:88–97, 2017. doi:10.1016/j.dam.2016.02.008.

48 Egon Wanke. k-NLC graphs and polynomial algorithms. *Discrete Applied Mathematics*, 54(2):251–266, 1994. doi:10.1016/0166-218X(94)90026-4.

Hamiltonicity Parameterized by Mim-Width Is (Indeed) Para-NP-Hard

Benjamin Bergougnoux
Aix-Marseille Université, CNRS, LIS, Marseille, France

Lars Jaffke
NHH Norwegian School of Economics, Bergen, Norway

Abstract

We prove that Hamiltonian Path and Hamiltonian Cycle are NP-hard on graphs of linear mim-width 26, even when a linear order of the input graph with mim-width 26 is provided together with input. This fills a gap left by a broken proof of the para-NP-hardness of Hamiltonicity problems parameterized by mim-width.

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

Keywords and phrases Hamiltonian Path, Hamiltonian Cycle, Mim-Width, Para-NP-Hardness

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.31

Related Version *Full Version*: https://arxiv.org/abs/2507.00612

Acknowledgements We thank Richard Wols for the counterexample presented in Section 2.1.

1 Introduction

Maximum induced matching width (mim-width) which was introduced in 2012 by Vatshelle [28] is by now a common staple among width parameters of graphs (e.g. [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]). One pillar in the understanding of its algorithmic use and limitations is that finding simply structured *induced* subgraphs tends to be tractable (solvable in XP time by the width of a given decomposition), while finding *non-induced* subgraphs is usually hard (NP-hard for constant mim-width). Most notably, finding long induced paths and cycles is in XP [6], while in [21], it has been claimed that Hamiltonian Cycle is NP-hard on graphs of linear mim-width 1. The latter claim rests on a proof that Hamiltonian Cycle is NP-hard on rooted directed path graphs given in [27], and in [21] it is shown that the graphs resulting from this reduction have linear mim-width 1. Unfortunately, as we show below, the reduction from [27] has a flaw, which means that the complexity of Hamiltonicity problems (or, more generally, problems of finding long non-induced paths or cycles) parameterized by mim-width remained open. In this paper, we repair this gap in the literature by proving the following result.

▶ **Theorem 1.** *The Hamiltonian Path and Hamiltonian Cycle problems are* NP*-hard on graphs of linear mim-width 26, even when a linear order of the input graph of mim-width* 26 *is provided as part of the input.*

Observe that hardness holds even when a linear order of small mim-width is provided at the input. Recently, Bergougnoux, Bonnet, and Duron [3] showed that computing the mim-width of a graph exactly is para-NP-hard as well. Note however that there might still be some f such that there is an XP time $f(k)$-approximation algorithm to mim-width k.

We would like to mention that the constant on the mim-width in our statements can likely be improved, and it is an interesting question where exactly the boundary between tractability and intractability lies for Hamiltonicity problems. Our results very much allow

© Benjamin Bergougnoux and Lars Jaffke;
licensed under Creative Commons License CC-BY 4.0

the possibility that Hamiltonicity is polynomial-time solvable on rooted directed path graphs or more generally, on graphs of mim-width 1. For Clique and a number of related problems, Otachi, Suzuki, and Tamura [26] recently showed that the boundary between tractable and intractable cases lies between mim-width 1 and 2. While Clique was known to be solvable in polynomial time on graphs of mim-width 1 (given a decomposition), they showed that it becomes NP-hard on graphs of mim-width 2. It would be interesting to establish a similar dividing line for Hamiltonicity problems.

Proofs of statements marked with "♠" are deferred to the full version.

2 Preliminaries

We denote the set of natural numbers by $\mathbb{N} = \{0, 1, \dots\}$, and for $n \in \mathbb{N}$, we use the shorthand $[n] = \{1, \dots, n\}$. All graphs considered in this paper are finite, undirected, and simple. For a graph G, we denote by $V(G)$ its vertices and by $E(G)$ its edges. For a vertex $v \in V(G)$, we denote its neighborhood by $N_G(v) = \{u \mid uv \in E(G)\}$. The *degree* of v is $|N_G(v)|$. For a set $X \subseteq V(G)$, the *subgraph of G induced by X*, denoted by $G[X]$, is the graph on vertex set X and edge set $\{uv \mid uv \in E(G), \{u, v\} \subseteq X\}$. A set $X \subseteq V(G)$ is *independent* if $G[X]$ has no edges and X is a *clique* if every pair of vertices in X is adjacent in G.

A *walk* in a graph G is a sequence of vertices $W = (v_1, \dots, v_r)$ such that for all $i \in [r-1]$, $v_i v_{i+1} \in E(G)$. If all vertices in W are pairwise distinct, then W is called a *path*, and if all vertices in $\{v_1, \dots, v_{r-1}\}$ are pairwise distinct and $v_1 = v_r$, then W is a *cycle*. A path/cycle is *Hamiltonian* if it contains all vertices of G.

A graph G is *bipartite* if there is a partition (A, B) of $V(G)$ such that A and B are independent. G is further called a *chain graph* if there is a linear order $a_1, \dots, a_n$ of A such that for all $i \in [n-1]$: $N(a_i) \subseteq N(a_{i+1})$.

Let G be a graph. Two distinct vertices $u, v \in V(G)$ are *false twins* if $N_G(u) = N_G(v)$ and $uv \notin E(G)$. Let $uv \in E(G)$. The *subdivision of uv* is the operation of replacing uv with a path (u, x, v) where x is a newly created vertex.

Linear Mim-width. Let G be a graph and $A \subseteq V(G)$. We let $\overline{A} = V(G) \setminus A$. We denote by $G[A, \overline{A}]$ the bipartite subgraph of G with vertex bipartition $(A, \overline{A})$ whose edges are $\{ab \mid ab \in E(G), a \in A, b \in \overline{A}\}$. A set of edges $M = \{a_i b_i \mid i \in [r]\} \subseteq E(G)$ is a *matching* if all vertices in $V(M) = \bigcup_{i \in [r]} \{a_i, b_i\}$ are pairwise distinct. A matching M is *induced* if $E(G[V(M)]) = M$. The *mim-value* of a set $A \subseteq V(G)$, denoted by $\mathsf{mim}_G(A)$, is the largest size of any induced matching in $G[A, \overline{A}]$. Let $\lambda = (v_1, \dots, v_n)$ be a linear order of $V(G)$. The *maximum induced matching width (mim-width) of λ* is $\max_{i \in [n]} \mathsf{mim}_G(\{v_1, \dots, v_i\})$.

The following observation will be helpful in a later proof.

▶ **Observation 2.** *Let H be a chain graph with bipartition (A, B). Then, $\mathsf{mim}_H(A) \leq 1$.*

2.1 Counterexample to the existing reduction

We recall the reduction from [27]. It is from Hamiltonian Cycle on bipartite graphs of maximum degree 3 and minimum degree 2 to Hamiltonian Cycle on rooted directed path graphs, which are intersection graphs of directed paths in a rooted tree.

Let G be a bipartite graph of maximum degree 3 and minimum degree 2, and let (A, B) be the vertex bipartition of G. We may assume that $|A| = |B| = n$, and fix arbitrary orderings on A and B, that is, we let $A = \{a_1, \dots, a_n\}$ and $B = \{b_1, \dots, b_n\}$ throughout. We construct a graph H as follows.

1. Let H be a copy of G.
2. Subdivide each edge $a_ib_j \in E(H)$ once. For convenience, we use a_ib_j to both denote the edge in G and the vertex in H created in the subdivision of this edge.
3. Make $\{a_ib_j \mid i, j \in [n], a_ib_j \in E(G)\}$ a clique in H.
4. For each $i \in [n]$, make a_i adjacent to all vertices $a_{i'}b_j \in V(H)$ where $1 \leq i' \leq i$, $j \in [n]$, $a_{i'}b_j \in E(G)$.
5. For each $i \in [n]$ such that b_i has degree 3 in G, add the vertex b'_i with $N_H(b'_i) = N_H(b_i)$ to H.

In [21] it is shown that the graph H as constructed above has linear mim-width 1.

In Figure 1, we show a bipartite graph G of maximum degree 3 and minimum degree 2 that has no Hamiltonian cycle, while the graph obtained from running the above reduction on input G has one, thus providing a counterexample to the proof in [27].

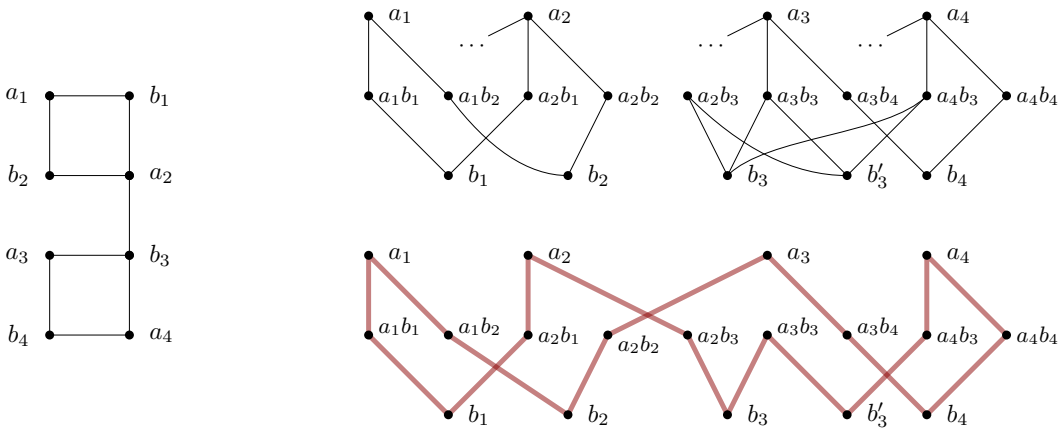


Figure 1 Counterexample to the construction from [27]. The three dots symbolize adjacency to all vertices to the left on the middle layer.

2.2 3-SAT

A 3-*CNF formula* is a Boolean formula in conjunctive normal form such that each clause has at most three pairwise distinct literals. Given a formula ϕ, we denote by $\mathsf{Var}(\phi)$ its set of variables. A truth assignment to the variables of ϕ is a a function $\alpha\colon \mathsf{Var}(\phi) \to \{0,1\}$. We say that a clause C is satisfied by α if C contains a literal ℓ with variable x such that $\alpha(x) = 1$ if $\ell = x$ and $\alpha(x) = 0$ if $\ell = \overline{x}$. Moreover, α is said to be a satisfying assignment of ϕ if it satisfies all the clauses of ϕ. We will reduce from the following problem.

3-SAT	
Input:	A 3-CNF formula ϕ.
Question:	Does ϕ admit a satisfying assignment?

3 Main result

In this section, we prove that the following problem is NP-hard for parameter value $w = 25$.

Hamiltonian Path/Linear Mim-Width	
Input:	A graph G and a linear order λ of $V(G)$.
Parameter:	The mim-width w of λ.
Question:	Does G have a Hamiltonian path?

We give a polynomial-time reduction from the 3-SAT problem. Let ϕ be a 3-CNF formula with variables $\mathsf{Var}(\phi) = \{x_1, \ldots, x_n\}$ and clauses $C_1, \ldots, C_m$.

Variable gadgets. For each $i \in [n]$, we create a variable gadget as follows. It consists of a sequence of m cycles on six vertices such that traversing each cycle clockwise corresponds to setting x_i to true, and traversing them counterclockwise corresponds to setting x_i to false. The construction ensures that for each variable, all cycles are traversed in the same way.

Let $i \in [n]$. We show how to construct the variable gadget $\mathtt{V}_i$. For each $j \in [m]$, it contains a cycle D_i^j whose vertices (listed in order) are called 0-$in(i,j)$, 0-$out(i,j)$, $z_{out}(i,j)$, 1-$out(i,j)$, 1-$in(i,j)$, $z_{in}(i,j)$. For each $j \in [m-1]$, we add an edge between 1-$out(i,j)$ and 1-$in(i,j+1)$, and between 0-$out(i,j)$ and 0-$in(i,j+1)$. See Figure 2 for an illustration. Moreover, there is a vertex s_i, adjacent to 0-$in(i,1)$ and 1-$in(i,1)$, and a vertex t_i, adjacent to 0-$out(i,m)$ and 1-$out(i,m)$.

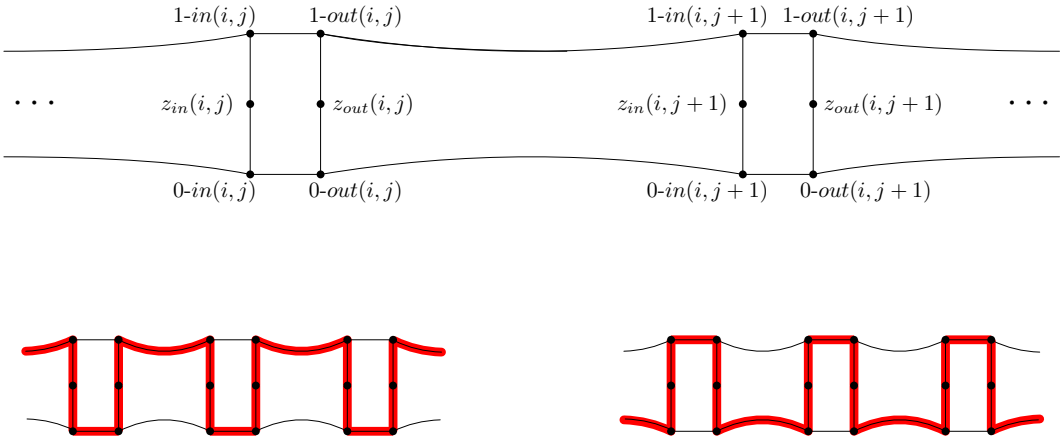


Figure 2 Illustration of the variable gadget. Bottom left (resp., right): a traversal of the sequence corresponding to setting the variable to false (resp., true).

Clause gadgets. For each clause C_j, we add a gadget $\mathtt{C}_j$ with the following property. Say variable x_i appears in C_j. If a given truth assignment to x_i satisfies C_j, then we can enter and collect the vertices of $\mathtt{C}_j$ from D_i^j (in the variable gadget $\mathtt{V}_i$) under the traversal of D_i^j corresponding to that truth assignment to x_i. To avoid "cheating", we need $\mathtt{C}_j$ to have the following property: once a path enters a vertex of $\mathtt{C}_j$, then we have to immediately collect all vertices from $\mathtt{C}_j$, and leave again via a prescribed exit, depending on where we entered $\mathtt{C}_j$. The following clause gadget due to Cygan, Kratsch, and Nederlof [14] achieves just that. We visualize these graphs in Figure 3.

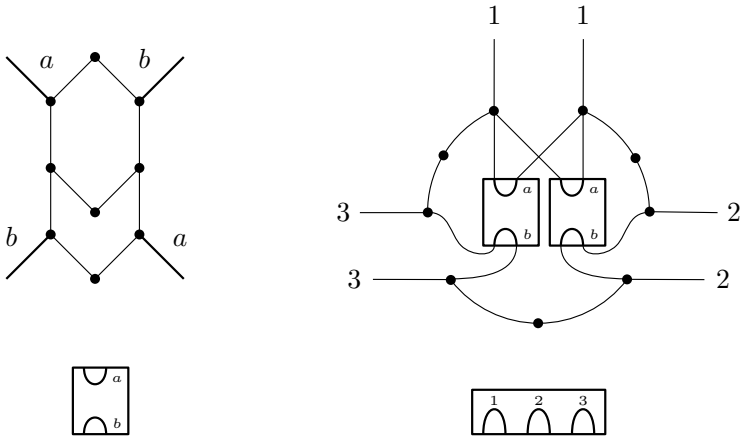


Figure 3 The clause gadgets due to Cygan, Kratsch, and Nederlof [14]. On the left, a gadget such that each Hamiltonian path that enters it via an edge labeled a (resp., b) must immediately collect all vertices in the gadget and leave via the other edge labeled a (resp, b). This gadget can be used for clauses of size two. On the right, a gadget for a clause of size three, with three entry-exit pairs and the analogous functionality.

▶ **Lemma 3** (Cygan, Kratsch, and Nederlof [14]). *For each $k \in \{2,3\}$, there is a graph Γ_k on at most 27 vertices with k distinguished pairs of unique vertices $\{(\sigma_1,\tau_1),\ldots,(\sigma_k,\tau_k)\}$ with the following property. Let G be a graph with an induced subgraph Γ_k, and let P be a Hamiltonian path of G. Then, $P \cap \Gamma_k$ is a (σ_i,τ_i)-path for some $i \in [k]$.*

Main construction. We now construct the graph G from the formula ϕ, see Figure 4 for an illustration. First, we add two vertices s and t to G, which will have degree one and are therefore the endpoints of any Hamiltonian path in G, if one exists. For each $i \in [n]$, we add a variable gadget $\mathtt{V}_i$. We furthermore add the edges ss_1, t_nt, and for all $i \in [n-1]$, we proceed as follows. We add a vertex p_{i+1} to G, and add the edges t_ip_{i+1} and $p_{i+1}s_{i+1}$.

Next, for each clause C_j with literals $\ell_1,\ldots,\ell_k$, we proceed as follows. We let the clause gadget $\mathtt{C}_j$ be a copy of Γ_k from Lemma 3; let (σ_1,τ_1), …, (σ_k,τ_k) be its distinguished vertices. For each $h \in [k]$, let x_{j_h} be the variable of literal ℓ_h. If x_{j_h} appears positively in C_j, then we add the edges $\{\sigma_h, 0\text{-}in(i,j)\}$ and $\{\tau_h, 0\text{-}out(i,j)\}$, and if x_{j_h} appears negated in C_j, then we add $\{\sigma_h, 1\text{-}in(i,j)\}$ and $\{\tau_h, 1\text{-}out(i,j)\}$. This finishes the main part of the construction.

Dummy edges. At this point, G may have arbitrarily high mim-width. To reduce the mim-width of the graph, we add lots of dummy edges. These edges are introduced in such a way that they decrease the mim-width without allowing to "cheat". That is, no Hamiltonian path of G will ever use any of these edges. We proceed as follows. For each $j \in [m-1]$, each $i \in [2..n]$, and each $h < i$, we add the edges $\{b_1\text{-}out(i,j), b_2\text{-}in(h,j+1)\}$, for any $b_1,b_2 \in \{0,1\}$. Moreover, for each $i,j \in [t]$ with $i < j$, we add the edge $\{t_i,p_j\}$.

This finishes the construction. Clearly, this reduction can be performed in polynomial time. We show that it is correct.

▶ **Lemma 4.** *If ϕ is satisfiable, then G has a Hamiltonian path.*

Proof. Suppose that ϕ has a satisfying assignment $\alpha\colon \mathsf{Var}(\phi) \to \{0,1\}$. We construct a Hamiltonian path P of G as follows. See Figure 5 for an illustration of this construction. First, P starts in s, s_1. If $\alpha(x_1)=1$, then the next vertex of P is $1\text{-}in(1,1)$, and if $\alpha(x_1)=0$, then the next vertex of P is $0\text{-}in(1,1)$. Suppose the former is the case, the latter case is symmetric.

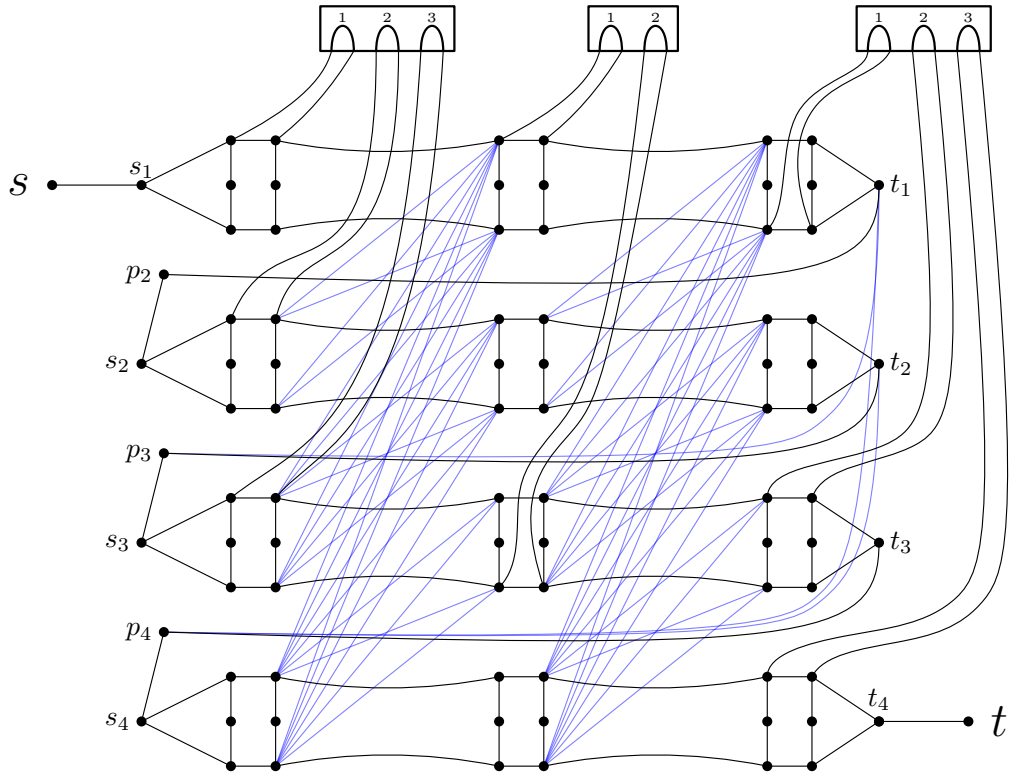


Figure 4 Overview of the construction. This is the example for the formula $(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_3 \vee x_4)$. The dummy edges are depicted in blue.

The next vertices on P are $z_{in}(1,1)$, 0-$in(1,1)$. Now, if x_1 appears positively in C_1, then by construction, we can enter the gadget $\mathtt{C}_1$ from 0-$in(1,1)$, collect all its vertices, and leave it to arrive at 0-$out(1,1)$. Note that in this case, x_1's truth assignment satisfied C_1. P then collects $z_{out}(1,1)$, 1-$out(1,1)$, and takes the edge to 1-$in(1,2)$. We proceed analogously in sequence on all D_1^j, until we reach t_1. Here we follow the edge to p_2, s_2, and proceed on $\mathtt{V}_2$ as we did on $\mathtt{V}_1$, now based on which value α assigns x_2. We repeat this process until we finally reach t.

Whenever the truth assignment $\alpha(x_i)$ satisfies a clause C_j, we are able to collect the vertices of $\mathtt{C}_j$. Therefore, the procedure described above indeed produces a Hamiltonian path of G. ◀

Let $s, t \in V(G)$, and let P be an (s,t)-path in G. For distinct $u, v \in V(P)$, we write $u \prec_P v$ if the path from s to v contains u. For two sets $X, Y \subseteq V(G)$, we write $X \prec_P Y$ if for all $x \in X$, $y \in Y$, $x \prec_P y$. For intuition on the following notions, recall Figure 2.

▶ **Definition 5.** *Let $a \in [n]$ and $b \in [m]$, and let P be a Hamiltonian path of G starting in s. We say that P* traverses $D_a^1, \ldots, D_a^b$ in true-order *if P contains the following as subpaths for every $j \in [b]$*
(P_{in}) 1-$in(a,j)$, $z_{in}(a,j)$, 0-$in(a,j)$
(P_{out}) 0-$out(a,j)$, $z_{out}(a,j)$, 1-$out(a,j)$
and for every $j \in [b-1]$, P contains the edge $\{1\text{-}out(i,j), 1\text{-}in(i,j+1)\}$. We say that P traverses $D_a^1, \ldots, D_a^b$ in false-order *if P contains the following as subpaths:*
(P_{in}) 0-$in(a,j)$, $z_{in}(a,j)$, 1-$in(a,j)$
(P_{out}) 1-$out(a,j)$, $z_{out}(a,j)$, 0-$out(a,j)$
and for every $j \in [b-1]$, P contains the edge $\{0\text{-}out(i,j), 0\text{-}in(i,j+1)\}$.

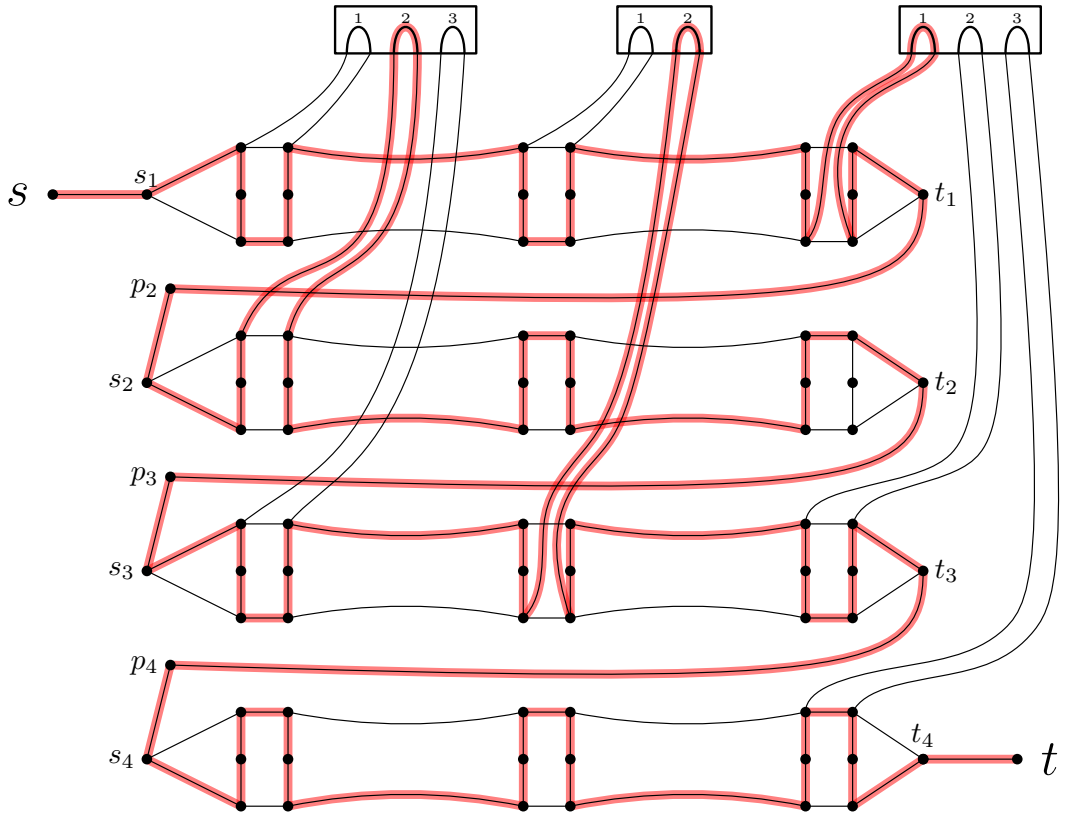


Figure 5 Example of the Hamiltonian path – of the graph presented in Figure 4 – constructed in the proof of Lemma 4 from the satisfying assignment α with $\alpha(x_1) = \alpha(x_3) = 0$ and $\alpha(x_2) = \alpha(x_4) = 1$. We omit the dummy edges to improve the legibility.

▶ **Definition 6.** *Let $a \in [n], b \in [m]$ and let P be a Hamiltonian (s,t)-path of G. We say that P is* (a,b)-respectable *if the following are satisfied.*

1. *For all $i \in [n]$ with $i < a$, we have $V(D_i^1) \prec_P V(D_i^2) \prec_P \cdots \prec_P V(D_i^m)$.*
2. *We have $V(D_a^1) \prec_P \cdots \prec_P V(D_a^b)$.*
3. *We have $V(\mathtt{V}_1) \prec_P \cdots \prec_P V(\mathtt{V}_{a-1}) \prec_P V(\mathtt{V}_a)$.*
4. *For every $i < a$, P traverses $D_i^1, \ldots, D_i^m$ in either true order or false order.*
5. *P traverses $D_a^1, \ldots, D_a^b$ in either true order or false order.*

▶ **Lemma 7 (♠).** *Let P be a Hamiltonian (s,t)-path of G. Then, P is (n,m)-respectable.*

▶ **Lemma 8.** *If G has a Hamiltonian path, then ϕ is satisfiable.*

Proof. Let P be a Hamiltonian path in G. By Lemma 7, P is (n,m)-respectable and in particular, for every $i \in [n]$, P traverses $D_i^1, \ldots, D_i^m$ in either true-order or false-order. We construct a truth assignment $\alpha \colon \mathsf{Var}(\phi) \to \{0,1\}$ as follows. For each $i \in [n]$, if P traverses $D_i^1, \ldots, D_i^m$ in true-order, we set $\alpha(x_i) = 1$, and if P traverses $D_i^1, \ldots, D_i^m$ in false-order, we set $\alpha(x_i) = 0$. Let $j \in [m]$. Since P is a Hamiltonian path, it collects the vertices in $\mathtt{C}_j$. By Lemma 3, there is some $i \in [n]$ such that P contains a path starting with a vertex in D_i^j, traversing all vertices in $\mathtt{C}_j$, and ending with a vertex in D_i^j. By construction, the two vertices in D_i^j are 0-$in(i,j)$, 0-$out(i,j)$, if x_i appears positively in C_j, and 1-$in(i,j)$, 1-$out(i,j)$, if x_i appears negated in C_j. Suppose without loss of generality that the former is the case. For P

to contain the described subpath, it must have entered D_i^j at vertex 1-$in(i,j)$, which implies that P traverses $D_i^1, \ldots, D_i^m$ in true-order. By our construction, $\alpha(x_i) = 1$, meaning that the literal of C_j containing x_i satisfies C_j. ◀

▶ **Lemma 9.** *We can construct in polynomial time a linear order on $V(G)$ of mim-width at most* 25*.*

Proof. Let λ be any linear order on $V(G)$ extending the following partial order:

$$\begin{aligned} & s, s_1, D_1^1, p_2, s_2, D_2^1, \ldots, p_n, s_n, D_n^1, V(\mathtt{C}_1), \ldots, \\ & \quad D_1^2, D_2^2, \ldots, D_n^2, V(\mathtt{C}_2), \\ & \quad \ldots, \\ & \quad D_1^{m-1}, D_2^{m-1}, \ldots, D_n^{m-1}, V(\mathtt{C}_{m-1}), \\ & \quad D_1^m, t_1, D_2^m, t_2, \ldots, D_n^m, t_n, t, V(\mathtt{C}_m) \end{aligned}$$

Throughout the following, we let $D = \bigcup_{(i,j)\in[n]\times[m]} D_i^j$, $P = \{p_i \mid i \in [n]\}$, $S = \{s\} \cup \{s_i \mid i \in [n]\}$, $T = \{t_i \mid i \in [n]\} \cup \{t\}$. Let $(A, \overline{A})$ be any cut induced by λ (such that A contains the first t vertices of λ for some $t \leq |V(G)|$), and let M be an induced matching in $G[A, \overline{A}]$. Each edge of M is one of the following types, and below we justify the bounds on their number indicated in the respective parenthesis.

1. Between $S \cup P$ and $S \cup T$ (≤ 1).
2. Between $S \cup T$ and D (≤ 1).
3. Between D and $V(\mathtt{C}_j)$, for some $j \in [m]$ (≤ 6).
4. Inside $V(\mathtt{C}_j)$ for some $j \in [m]$ (≤ 13).
5. Inside D (≤ 4).

For edges of type 1, note that $G[A \cap (S \cup P), \overline{A} \cup (S \cup T)]$ is a chain graph plus possibly one edge $p_i s_i$, for some $i \in [n]$. Recall by Observation 2, M can contain at most one edge from any chain graph. Furthermore, if $p_i s_i \in M$, then M has no other edge from this subgraph, since in this case, $p_i \in A$ is adjacent to all vertices in $T \cap \overline{A}$ that have a neighbor in $(S \cup P) \cap A$. So in either case, there is at most one edge of type 1 in M.

For edges of type 2, such edges can only occur if either some s_i is the last vertex of A or if some t_i is the first vertex of $\overline{A}$; in any case, there is at most one such edge in M. The number of edges of types 3 and 4 are trivial upper bounds.

Lastly, consider edges of type 5. Let (j, i) be the lexicographically largest pair such that A contains vertices from D_i^j. First, there are at most two edges from D_i^j itself in M. Next, consider the edges in M that are between $D^j = \bigcup_{i\in[n]} D_i^j$ and $D^{j+1} = \bigcup_{i\in[n]} D_i^{j+1}$. Suppose there is some $h \in [n]$ such that there is an edge $d_j d_{j+1}$ in M, where $d_j \in D_h^j$ and $d_{j+1} \in D_h^{j+1}$. Note that there are at most two such edges per h. Moreover, in this case, M has no other edges between D^j and D^{j+1}, since for each such edge, at least one endpoint has a neighbor in $\{d_j, d_{j+1}\}$. So in this case, there are at most four edges of type 5 in total. If there is no such h, then by a similar argument, M contains at most one edge between D^j and D^{j+1}, in which case we have at most three edges of type 5. ◀

This finishes the proof for Hamiltonian Path. By adding another degree-two vertex adjacent to only s and t, we also obtain the same result for the Hamiltonian Cycle problem and the fact that adding a vertex to a graph increase its mim-width by at most 1.

▶ **Theorem 1.** *The Hamiltonian Path and Hamiltonian Cycle problems are* NP*-hard on graphs of linear mim-width 26, even when a linear order of the input graph of mim-width* 26 *is provided as part of the input.*

References

1 Brage I. K. Bakkane and Lars Jaffke. On the hardness of generalized domination problems parameterized by mim-width. In Holger Dell and Jesper Nederlof, editors, *Proceedings of the 17th International Symposium on Parameterized and Exact Computation, IPEC 2022*, volume 249 of *LIPIcs*, pages 3:1–3:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.IPEC.2022.3.

2 Rémy Belmonte and Martin Vatshelle. Graph classes with structured neighborhoods and algorithmic applications. *Theor. Comput. Sci.*, 511:54–65, 2013. doi:10.1016/J.TCS.2013.01.011.

3 Benjamin Bergougnoux, Édouard Bonnet, and Julien Duron. Mim-width is paranp-complete. In Keren Censor-Hillel, Fabrizio Grandoni, Joël Ouaknine, and Gabriele Puppis, editors, *Proceedings of the 52nd International Colloquium on Automata, Languages, and Programming, ICALP 2025*, volume 334 of *LIPIcs*, pages 25:1–25:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPICS.ICALP.2025.25.

4 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 34th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2023)*, pages 3282–3304. SIAM, 2023. doi:10.1137/1.9781611977554.ch125.

5 Benjamin Bergougnoux, Thekla Hamm, Lars Jaffke, and Paloma T. Lima. On algorithmic applications of $\mathcal{F}$-branchwidth. In *ESA 2025*, 2025. To appear.

6 Benjamin Bergougnoux and Mamadou Moustapha Kanté. More applications of the d-neighbor equivalence: Acyclicity and connectivity constraints. *SIAM J. Discret. Math.*, 35(3):1881–1926, 2021. doi:10.1137/20M1350571.

7 Benjamin Bergougnoux, Charis Papadopoulos, and Jan Arne Telle. Node multiway cut and subset feedback vertex set on graphs of bounded mim-width. *Algorithmica*, 84(5):1385–1417, 2022. doi:10.1007/S00453-022-00936-W.

8 Hans L. Bodlaender, Carla Groenland, Hugo Jacob, Lars Jaffke, and Paloma T. Lima. XNLP-Completeness for parameterized problems on graphs with a linear structure. *Algorithmica*, 87(4):465–506, 2025. doi:10.1007/S00453-024-01274-9.

9 Flavia Bonomo-Braberman, Nick Brettell, Andrea Munaro, and Daniël Paulusma. Solving problems on generalized convex graphs via mim-width. *J. Comput. Syst. Sci.*, 140:103493, 2024. doi:10.1016/J.JCSS.2023.103493.

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 Nick Brettell, Andrea Munaro, Daniël Paulusma, and Shizhou Yang. Comparing width parameters on graph classes. *Eur. J. Comb.*, 127:104163, 2025. doi:10.1016/J.EJC.2025.104163.

13 Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. Fast dynamic programming for locally checkable vertex subset and vertex partitioning problems. *Theor. Comput. Sci.*, 511:66–76, 2013. doi:10.1016/J.TCS.2013.01.009.

14 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.

15 Eduard Eiben, Robert Ganian, Thekla Hamm, Lars Jaffke, and O-joung Kwon. A unifying framework for characterizing and computing width measures. In Mark Braverman, editor, *Proceedings of the 13th Innovations in Theoretical Computer Science Conference, ITCS 2022*, volume 215 of *LIPIcs*, pages 63:1–63:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ITCS.2022.63.

16 Esther Galby, Andrea Munaro, and Bernard Ries. Semitotal domination: New hardness results and a polynomial-time algorithm for graphs of bounded mim-width. *Theor. Comput. Sci.*, 814:28–48, 2020. doi:10.1016/J.TCS.2020.01.007.

17 Petr A. Golovach, Pinar Heggernes, Mamadou Moustapha Kanté, Dieter Kratsch, Sigve Hortemo Sæther, and Yngve Villanger. Output-polynomial enumeration on graphs of bounded (local) linear mim-width. *Algorithmica*, 80(2):714–741, 2018. doi:10.1007/S00453-017-0289-1.

18 Carolina Lucía Gonzalez and Felix Mann. On d-stable locally checkable problems parameterized by mim-width. *Discret. Appl. Math.*, 347:1–22, 2024. doi:10.1016/J.DAM.2023.12.015.

19 Svein Høgemo, Jan Arne Telle, and Erlend Raa Vågset. Linear mim-width of trees. In Ignasi Sau and Dimitrios M. Thilikos, editors, *Proceedings of the 45th International Workshop on Graph-Theoretic Concepts in Computer Science, WG 2019*, volume 11789 of *Lecture Notes in Computer Science*, pages 218–231. Springer, 2019. doi:10.1007/978-3-030-30786-8_17.

20 Lars Jaffke, O-joung Kwon, Torstein J. F. Strømme, and Jan Arne Telle. Mim-width III. Graph powers and generalized distance domination problems. *Theor. Comput. Sci.*, 796:216–236, 2019. doi:10.1016/J.TCS.2019.09.012.

21 Lars Jaffke, O-joung Kwon, and Jan Arne Telle. Mim-Width I. Induced path problems. *Discret. Appl. Math.*, 278:153–168, 2020. doi:10.1016/J.DAM.2019.06.026.

22 Lars Jaffke, O-joung Kwon, and Jan Arne Telle. Mim-width II. The feedback vertex set problem. *Algorithmica*, 82(1):118–145, 2020. doi:10.1007/S00453-019-00607-3.

23 Lars Jaffke, O-joung Kwon, and Jan Arne Telle. Classes of intersection digraphs with good algorithmic properties. *J. Graph Theory*, 106(1):110–148, 2024. doi:10.1002/JGT.23065.

24 Stefan Mengel. Lower bounds on the mim-width of some graph classes. *Discret. Appl. Math.*, 248:28–32, 2018. doi:10.1016/J.DAM.2017.04.043.

25 Andrea Munaro and Shizhou Yang. On algorithmic applications of sim-width and mim-width of (H_1, H_2)-free graphs. *Theor. Comput. Sci.*, 955:113825, 2023. doi:10.1016/J.TCS.2023.113825.

26 Yota Otachi, Akira Suzuki, and Yuma Tamura. Finding induced subgraphs from graphs with small mim-width. In Hans L. Bodlaender, editor, *Proceedings of the 19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024)*, volume 294 of *LIPIcs*, pages 38:1–38:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.SWAT.2024.38.

27 B. S. Panda and Dinabandhu Pradhan. NP-Completeness of Hamiltonian cycle problem on rooted directed path graphs. *CoRR*, abs/0809.2443, 2008. arXiv:0809.2443.

28 Martin Vatshelle. *New Width Parameters of Graphs.* PhD thesis, University of Bergen, Norway, 2012.

The PACE 2025 Parameterized Algorithms and Computational Experiments Challenge: Dominating Set and Hitting Set

Mario Grobler
University of Bremen, Germany

Sebastian Siebertz
University of Bremen, Germany

Abstract

The 10th iteration of the of the Parameterized Algorithms and Computational Experiments challenge (PACE) 2025 was devoted to engineer algorithms solving the Dominating Set problem as well as the Hitting Set problem. In contrast to the last iterations, these problems are (under standard assumptions) *not* fixed-parameter tractable (fpt) in general. However, restricting the structure of the input (e.g. to planar graphs or degenerate graphs for Dominating Set, or to set systems with sets of bounded size for Hitting Set) renders these problems fpt. Following the spirit of the last iterations of the PACE challenge, there is an exact track and a heuristic track for each problem; each track coming with a benchmark set of 100 public instances and 100 private instances. Overall, the PACE 2025 had 71 participants from 25 teams, 13 countries, and 3 continents. In this report, we briefly describe the setup of the challenge, the selection of benchmark instances, as well as the ranking of the participating teams. We also briefly outline the approaches used in the submitted solvers.

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

Keywords and phrases PACE 2025 Report, Dominating Set, Hitting Set, Algorithm Engineering, FPT, Heuristics

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.32

Acknowledgements The PACE challenge was supported by Networks [28]. The prize money (€4000) was generously provided by Networks [28], an NWO Gravitation project of the University of Amsterdam, Eindhoven University of Technology, Leiden University and the Center for Mathematics and Computer Science (CWI). We are grateful to the whole optil.io team, especially to Jan Badura for the fruitful collaboration and for hosting the competition at the optil.io online judge system. Further, this work was supported by the Data Science Center of the University of Bremen (DSC@UB) funded by the State of Bremen.

1 Introduction

The Parameterized Algorithms and Computational Experiments (PACE) Challenge, now in its 10th iteration, continues its mission of advancing the practical impact of parameterized and fine-grained algorithm design. Since its inception in PACE 2016, which focused on Treewidth and Feedback Vertex Set, the challenge has presented a diverse sequence of computational problems, each selected for its theoretical richness and relevance to real-world algorithm engineering.

© Mario Grobler and Sebastian Siebertz;
licensed under Creative Commons License CC-BY 4.0

PACE 2016	Treewidth and Feedback Vertex Set	[12];
PACE 2017	Treewidth and Minimum Fill-In	[13];
PACE 2018	Steiner Tree	[8];
PACE 2019	Vertex Cover and Hyper-Treewidth	[15];
PACE 2020	Treedepth	[26];
PACE 2021	Cluster Editing	[24];
PACE 2022	Directed Feedback Vertex Set	[19];
PACE 2023	Twinwidth	[4];
PACE 2024	One-Sided Crossing Minimization	[25].

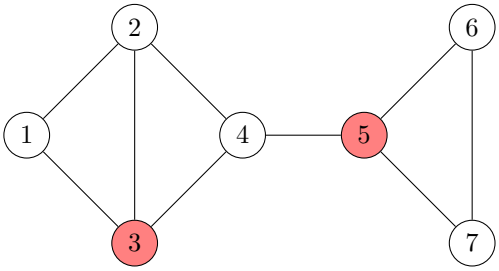
The overarching goal of PACE 2025 remains the same: to rigorously evaluate state-of-the-art algorithms for NP-complete problems, blending provable algorithmic guarantees with thorough experimental evaluation. This year's chosen problems, Dominating Set and Hitting Set, are canonical in complexity theory, embodying both theoretical depth and experimental challenge.

2 This year's problems

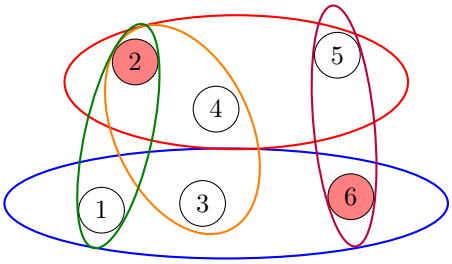
We briefly introduce the notation used. A *graph* G consists of its non-empty vertex set $V(G)$ and its edge set $E(G) \subseteq \binom{V(G)}{2}$. In particular, all graphs in this report are undirected, simple, and without loops. We define the (open) neighborhood of a vertex $v \in V(G)$ by $N(v) = \{u \in V(G) \mid \{u, v\} \in E(G)\}$. Similarly, we define the closed neighborhood of v by $N[v] = N(v) \cup \{v\}$. A *set system* $\mathcal{S}$ consists of its non-empty universe $V(\mathcal{S})$ and its set of sets $E(\mathcal{S}) \subseteq 2^{V(\mathcal{S})}$.

The Dominating Set and Hitting Set problems are classical NP-complete problems. In particular, Hitting Set was one of Karp's 21 NP-complete problems [23], and Dominating Set is NP-complete via its immediate equivalence to Set Cover [18], the dual of Hitting Set.

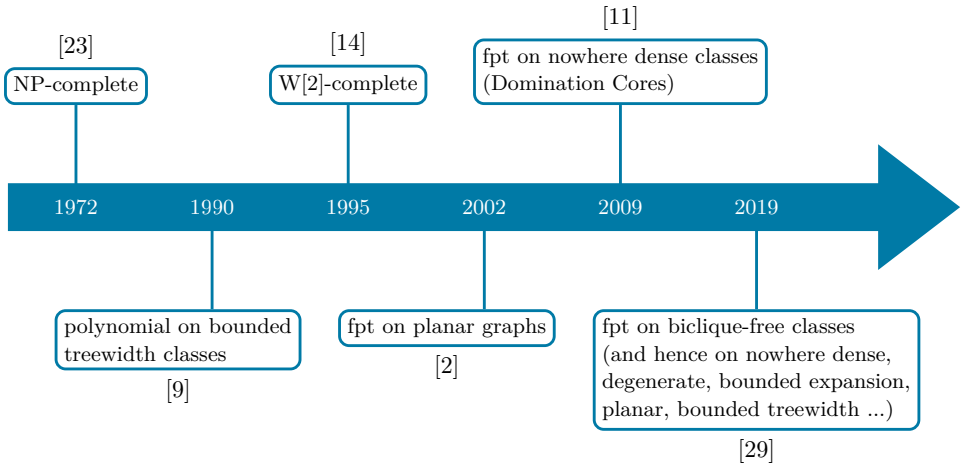
An instance of Dominating Set consists of an (undirected) graph G and an integer k, and the goal is to decide whether D admits a dominating set of size at most k, that is, a set $D \subseteq V(G)$ with $|D| \leq k$ such that $\bigcup_{v \in D} N[v] = V(G)$. Dominating Set is an important NP-complete problem known to be W[2]-complete in general [14] but fixed-parameter tractable on several restricted graph classes, see e.g. [2, 9, 11, 29]. In general, there is an algorithm that, for each k, counts the number of dominating sets of size k in $\mathcal{O}(1.5048^n)$ time [32].



(a) A dominating set.

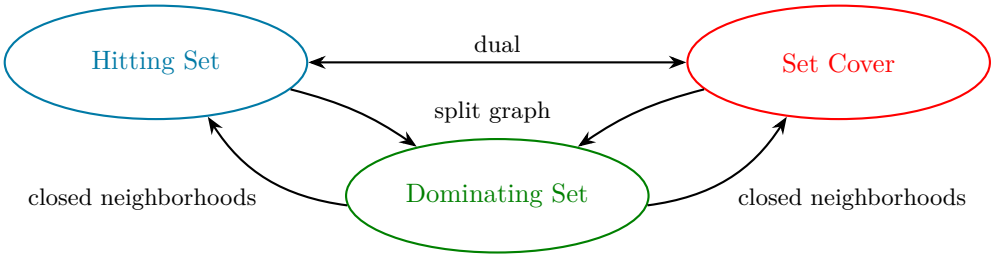


(b) A hitting set.



An instance of Hitting Set consists of a set system $\mathcal{S}$ and an integer k, and the goal is to decide whether $\mathcal{S}$ admits a hitting set of size at most k, that is, a set $H \subseteq V(\mathcal{S})$ such that every set $S \in E(\mathcal{S})$ contains at least one vertex from H, that is, for all $S \in E(\mathcal{S})$ we have $S \cap H \neq \varnothing$. Hitting Set generalizes problems like Vertex Cover, Feedback Vertex Set, Dominating Set, and many other problems. It is both NP-complete [23] and W[2]-complete [14], and yet tractable under certain structural restrictions. For example, for 3-Hitting Set (i.e., Hitting Set where every set has cardinality 3), the problem is fpt; the currently fastest known algorithm is due to Tsur [31], yielding a runtime of $\mathcal{O}^*(2.0409^k)$ (where k is the solution size), which generalizes to an $\mathcal{O}^*(d-1+0.0409^k)$-time algorithm for d-Hitting Set (i.e., Hitting Set where every set has cardinality d), which is faster for $d \leq 6$ compared to the the algorithm of Fernau [17].

An instance of Set Cover consists of a set system $\mathcal{S}$ and an integer k, and the goal is to decide whether $\mathcal{S}$ admits a set cover of size at most k, that is, a subset $C \subseteq E(\mathcal{S})$ such that $\bigcup_{S \in C} S = V(\mathcal{S})$. Hitting Set, Dominating Set, and Set Cover are easily seen to be equivalent by standard polynomial-time reductions.



First, Set Cover and Hitting Set are equivalent: one can construct a Hitting Set instance from a Set Cover instance simply by swapping the roles of elements and sets, and vice versa, ensuring that feasible solutions correspond one-to-one. Second, Dominating Set reduces to Set Cover via the reduction that takes the universe to be the vertices of the graph and, for each vertex, introduces a subset consisting of its closed neighborhood. Then dominating sets correspond directly to set covers (of identical size). Finally, given a Set Cover instance $\mathcal{S}$, build the split graph $G(\mathcal{S})$ with vertex set $V(\mathcal{S}) \cup E(\mathcal{S})$ where $E(\mathcal{S})$ induces a clique, $V(\mathcal{S})$ is an independent set, and $S \in E(\mathcal{S})$ is adjacent to $u \in V(\mathcal{S})$ iff $u \in S$. Then dominating sets of $G(\mathcal{S})$ may be assumed to be subsets of $V(\mathcal{S})$ and correspond one-to-one to set covers of $\mathcal{S}$. Note that these correspondences are L-reductions (linear

reductions), which also preserve approximation properties. Dominating Set, Hitting Set, and Set Cover parameterized by solution size are among the most frequently cited W[2]-complete problems in parameterized complexity [14].

3 The Setup of PACE 2025

The PACE 2025 competition followed the established multi-track format. The exact track demanded optimal solutions within strict resource limits, while the heuristic track emphasized solution quality under time pressure. Rigor in correctness was paramount: a small test set and verifier were released early, and a verification phase ensured solver reliability.

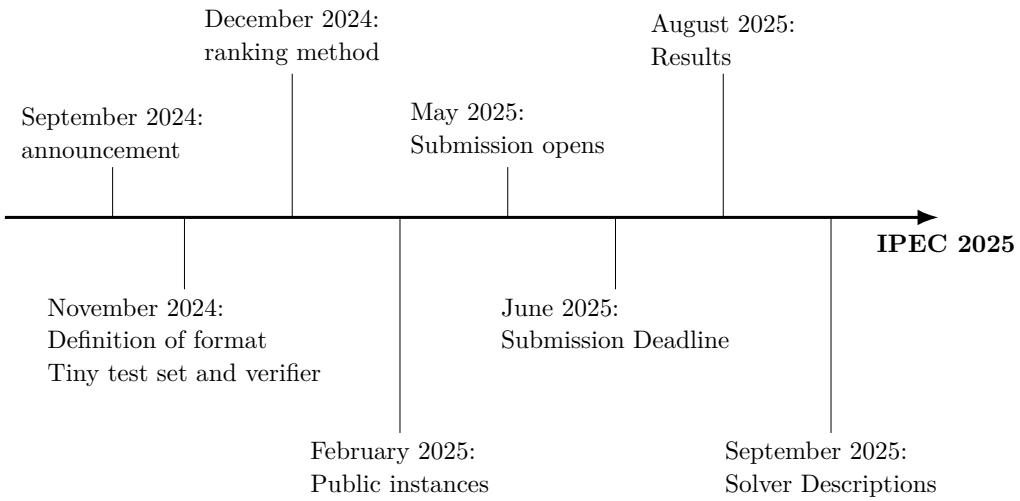


Figure 2 The timeline of PACE 2025.

3.1 Exact Tracks

Each exact track of PACE 2025 consisted of 200 benchmark instances, split into 100 public and 100 private instances. While the public instances were released for solver development and testing, the official evaluation was performed solely on the private set. Each solver was given a time limit of 30 minutes per instance, with an additional mercy time of 25 seconds, and was executed under a memory cap of 16 GB RAM, increased from 8 GB in previous years. To ensure reliability, an additional verification phase was conducted to verify correctness of submitted solvers. As in earlier editions, only solvers implementing provably correct algorithms were permitted, although formal proofs of correctness were not required. The winner of each track was determined by the number of solved private instances; in case of ties, the cumulative running time served as the tie-breaker.

3.2 Heuristic Tracks

The heuristic tracks of PACE 2025 followed a setup similar to the exact tracks, with 200 benchmark instances in total, split into 100 public and 100 private instances. Each solver was given a time limit of 5 minutes per instance, after which a SIGTERM signal was sent, followed by an additional 20 seconds mercy time before a final SIGKILL was sent. The mercy time was primarily intended to ensure that solvers had a sufficient amount of time

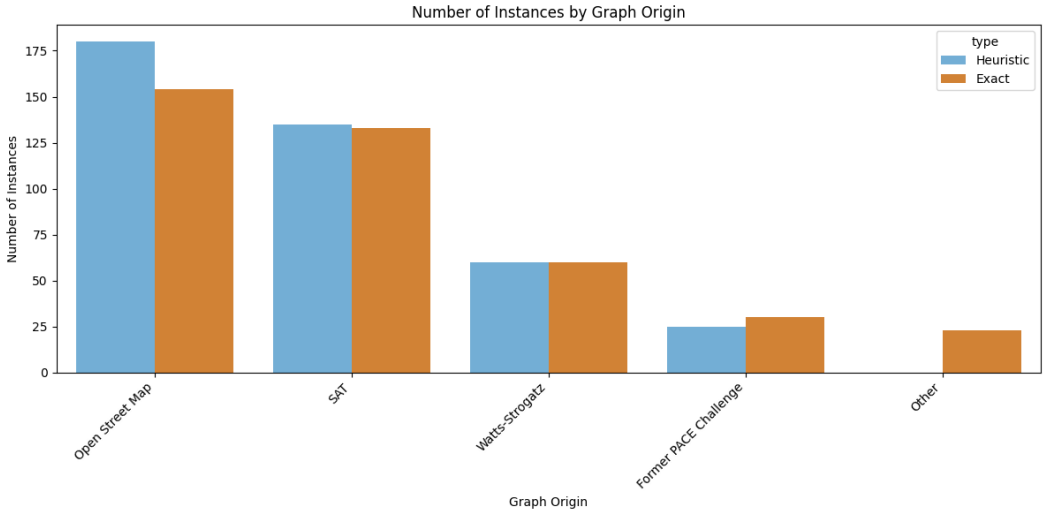
to write their output after termination was requested. As in the exact tracks, each solver was executed under a memory cap of 16 GB RAM, increased from 8 GB in previous years. Submissions were ranked by the sum over all instances of the scores computed by the function $f(k) = \left(\frac{u-k}{u-k^*}\right)^2$, where $u = \min\{n, 2k^*\}$, k^* is the best known solution value for the instance and k is the produced solution value. Hence, improving already good solutions was more beneficial than improving bad solutions, and instances where k^* is much smaller than n did not automatically yield a score of close-to-one.

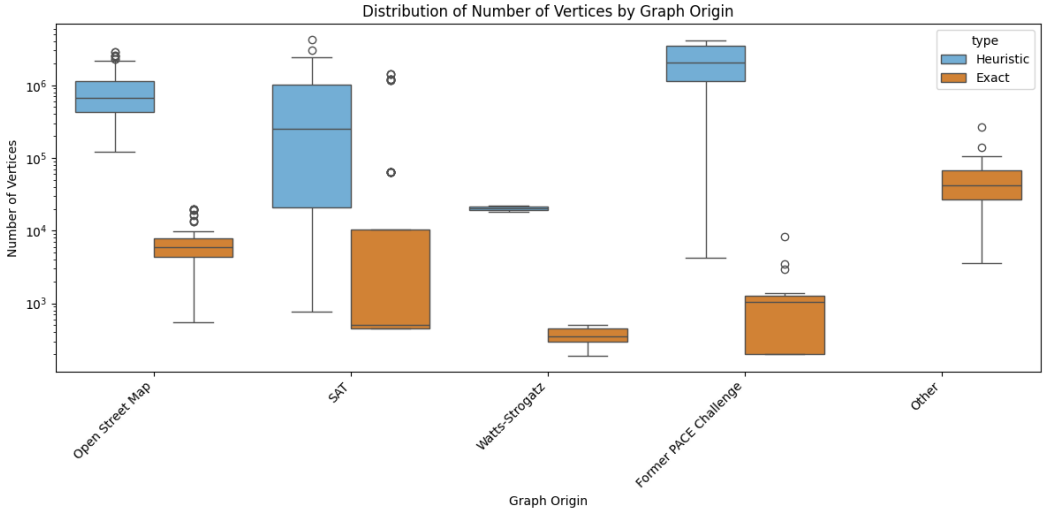
3.3 Benchmark Set

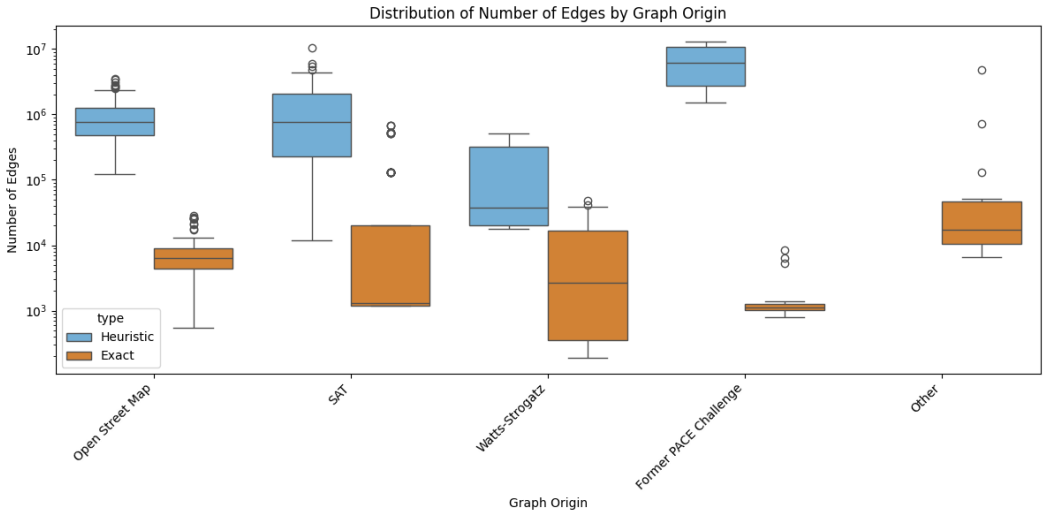
The benchmark set of PACE 2025 was selected with the objective of providing instances with nice structural properties, rather than relying solely on random or synthetic constructions. The collection was drawn from a diverse set of sources:

- **OpenStreetMap (OSM)**. These instances are (nearly) planar graphs derived from road networks. Interestingly, many solvers were able to compute exact solutions very efficiently on such instances, even for comparatively large graphs.
- **SATLIB**. Graph instances obtained from reductions of SATLIB benchmarks [21], which inherit structural features of the underlying SAT formulas. Again, a large fraction of solvers was capable of producing exact solutions within short running times
- **Watts-Strogatz random graphs**. The Watts-Strogatz model generates graphs by starting from a regular ring lattice and randomly rewiring edges with a given probability, thereby interpolating between regular and small-world structures. In contrast to the previous categories, these instances proved particularly challenging: even relatively small graphs remained intractable for most solvers.
- **Previous PACE challenges.** A small subset of instances was reused from earlier editions, in particular from PACE 2019 (Vertex Cover) and PACE 2021 (Cluster Editing).
- **Other.** The remaining instances include e.g. social network graphs derived from real-world social networks, such as Facebook. Despite their large size, many of these graphs could be solved quickly, with optimal solution sizes typically being very small. Other instances orginate from action sequences or were obtained by taking disjoint unions of the aforementioned instances with additional random edges between the components.

The benchmark set was employed uniformly across both Dominating Set and Hitting Set, and across both exact and heuristic tracks. For the heuristic tracks, larger instances were selected. Where necessary, appropriate reductions were applied to adapt instances to the respective problem formulations. For example, SAT instances were reduced to Dominating Set or Hitting Set, either directly or via intermediate reductions such as Vertex Cover. This ensured that structurally rich instances could be shared across problems and tracks. We note that the public benchmark set did not include any Watts-Strogatz instances. All other categories were approximately balanced between public and private sets. As the other categories turned out to be quite easy for many solvers, there was a high chance that many teams would achieve perfect scores on the public instances. As Watts-Strogatz graphs turned out to be considerably more difficult, they were eventually incorporated into the private test set. Importantly, in the exact tracks of both Dominating Set and Hitting Set, every non-Watts-Strogatz instance was solved by at least one solver. The only unsolved cases were precisely the Watts-Strogatz instances: 14 in the Dominating Set track and 15 in the Hitting Set track, underlining their role as the most challenging instances.


Number of Instances by Graph Origin
Number of Instances
0
25
50
75
100
125
150
175
type
Heuristic
Exact
Open Street Map
SAT
Watts-Strogatz
Former PACE Challenge
Other
Graph Origin


Distribution of Number of Vertices by Graph Origin
Number of Vertices
10^3
10^4
10^5
10^6
type
Heuristic
Exact
Open Street Map
SAT
Watts-Strogatz
Former PACE Challenge
Other
Graph Origin


Distribution of Number of Edges by Graph Origin
Number of Edges
10^3
10^4
10^5
10^6
10^7
type
Heuristic
Exact
Open Street Map
SAT
Watts-Strogatz
Former PACE Challenge
Other
Graph Origin

3.4 Verification Phase and Verification Set

For the first time in the history of the PACE challenge, a dedicated verification phase was introduced. In this phase, all submitted solvers were made publicly available to the community, and participants were explicitly invited to test each other's implementations in order to identify possible bugs. Moreover, all solvers were evaluated against a large verification dataset, which was not used for scoring but served as an additional safeguard: solvers that produced incorrect solutions on any of these instances were allowed to be repaired in case of minor fixes and otherwise disqualified from the competition.

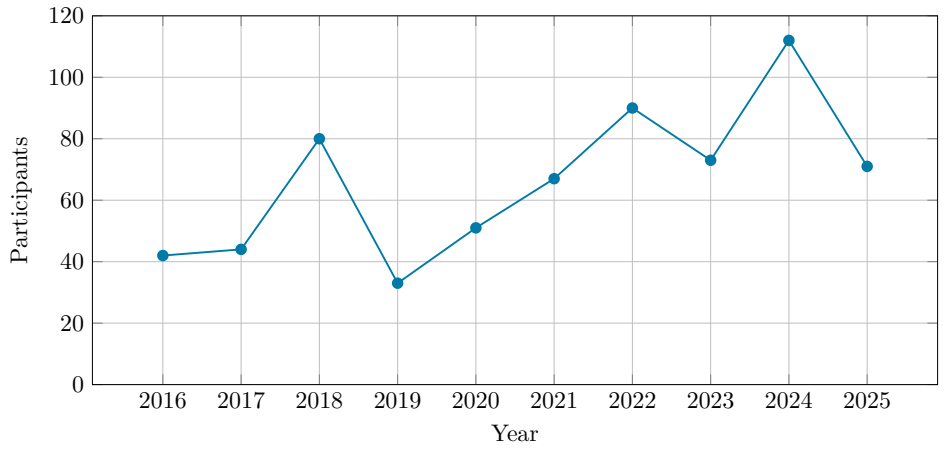
The verification set included large numbers of random graphs generated under different generative models, such as the Erdős-Rényi model [16], the Barabási-Albert model [5], the Watts-Strogatz model [33], and power-law cluster graphs following the algorithm of Holme and Kim [20], as well as uniform intersection graphs and several additional random graph models. In addition, the dataset contained named graphs such as grids, trees, ladders, cliques, paths, and wheels. Finally, a randomly selected subset of the large-scale STRIDE dataset (Sharing and Testing Repository for Instance Deployment and Evaluation, available at `https://domset.algorithm.engineering/`), which is maintained by Manuel Penschuck.

3.5 Evaluation Infrastructure

For baseline testing, the organizers employed an internal solver combining a set of problem-specific reduction rules with the commercial ILP solver CPLEX [10]. For both problems, dedicated verifiers were made available to participants; these were implemented as Java/Maven projects and ensured the correctness of submitted solutions (however, optimality was not checked). All solvers were executed inside Docker containers, using a common template that was also released publicly. Each container provided an identical environment, based on Ubuntu with 16 GB RAM and a single CPU core, thereby guaranteeing portability and reproducibility. This homogeneous setup allowed all solvers to be executed seamlessly on the compute cluster of the Data Science Center at the University of Bremen (`https://www.dsc-ub.de/`), ensuring fair and consistent evaluation across teams.

4 Participants

PACE 2025 had 71 participants from 25 teams, 13 countries, and 3 continents. We list the results below. A graduation cap (🎓) indicates a team where students are the main contributors.



5 Results

5.1 Dominating Set – Exact Track

5.1.1 Ranking

Rank	Team	Score	Time	Link
1	**UzL** Max Bannach, Florian Chudigiewitsch, Marcel Wienöbst *European Space Agency, University of Lübeck*	81	46111 s	
2	**Bad Dominating Set Maker** Alexander Dobler, Simon D. Fink, Mathis Rocton *TU Wien*	80	50253 s	
3	**OBLX** Jona Dirks, Enna Gerhard, Victoria Kaial, Lucas Lorieau *Université Clermont-Auvergne, University of Bremen*	80	64579 s	
4	**Alpaca** Tobias Röhr *Hasso Plattner Institute, University of Potsdam*	76	50057 s	
5	**Shadoks** Guilherme D. da Fonseca, Fabien Feschet, Yan Gerard *Aix-Marseille Université, Université Clermont-Auvergne*	73	65713 s	
6	**PaceYourself** Lukas Geis, Alexander Leonhardt, Johannes Meintrup, Ulrich Meyer, Manuel Penschuck *Goethe University Frankfurt, Technische Hochschule Mittelhessen*	71	83392 s	
7	**Swats** Sylwester Swat *Poznań University of Technology*	64	70325 s	
8	**Floris** Floris van der Hout *Utrecht University*	62	92309 s	
9	**AEG Heidelberg** Adil Chhabra, Marlon Dittes, Ernestine Großmann, Kenneth Langedal, Henrik Reinstädtler, Christian Schulz, Darren Strash, and Henning Woydt *Heidelberg University, Hamilton College*	48	107050 s	
10	**DSHunter** Paweł Putra *University of Warsaw*	22	142228 s	
11	**fu2025_pace** Yi-Hui Fan, Arthur Gisbrecht, Rashid Harvey, Vinzent Jörß, Bernadette Keßler, Oliver Seitz *FU Berlin*	19	163971 s	

12	**Elias & Thor** 🎓 Elias Rasmussen, Thor Vejen Eriksen *University of Copenhagen*	18	156974 s	
DQ	**Marina, Viktoriia & Fedor** 🎓 Marina Fridman, Viktoriia Krivogornitsyna, Fedor Kurmazov *Saint Petersburg State University of Economics*			

5.1.2 Short Description of the Dominating Set Exact Track Winning Solvers

For the **UzL** solver and the **Bad Dominating Set Maker**, we refer to Section 5.3.2, as these solvers are also the winners of the Hitting Set Exact Track, and there are only minor differences in the strategies employed for Dominating Set and Hitting Set.

The **OBLX** solver uses an extended version of Dominating Set and applies a large set of reduction rules, both well-known rules and rules especially designed for planar graphs. It then reduces the resulting instance to Hitting Set and further to weighted Max-Sat, which is then solved using the `EvalMaxSat` solver [3]. Using the result, a postprocessing routine to revert the modifications of the previously applied reduction rules is applied on the solution. This step is necessary in order to obtain the solution for the original Dominating Set instance.

5.2 Dominating Set – Heuristic Track

5.2.1 Ranking

Rank	Team	Score	Time	Link
1	**Florian & Guillaume** Florian Fontan, Guillaume Verger	99.81	28553 s	
2	**Root** Canhui Luo, Zhipeng Lv, Zhouxing Su, Qingyun Zhang *Huazhong University of Science and Technology*	99.64	30035 s	
3	**Swats** Sylwester Swat *Poznań University of Technology*	99.35	22062 s	
4	**Shadoks** Guilherme D. da Fonseca, Fabien Feschet, Yan Gerard *Aix-Marseille Université, Université Clermont-Auvergne*	99.02	31545 s	
5	**AEG Heidelberg** Adil Chhabra, Marlon Dittes, Ernestine Großmann, Kenneth Langedal, Henrik Reinstädtler, Christian Schulz, Darren Strash, and Henning Woydt *Heidelberg University, Hamilton College*	98.06	30014 s	

6	**PaceYourself** Lukas Geis, Alexander Leonhardt, Johannes Meintrup, Ulrich Meyer, Manuel Penschuck *Goethe University Frankfurt, Technische Hochschule Mittelhessen*	96.92	30024 s	
7	**Samuel** 🎓 Samuel Füßinger *Eberhard Karls Universität Tübingen*	96.80	30017 s	
8	**Viacheslav** 🎓 Viacheslav Khrushchev *HSE University Moscow*	96.27	30024 s	
9	**Hui, Bo, Yexin & Xinyun** 🎓 Hui Kong, Bo Peng, Yexin Peng, Xinyun Wu *Hubei University of Technology, Southwestern University of Finance and Economics*	96.16	30025 s	
10	**Greeduce** Adam Polak, Jonas Schmidt *Bocconi University*	95.82	30016 s	
11	**Aslam, Ashwin, Sahil, Nithin, Pankaj & Edwin** 🎓 Aslam Ameen, Ashwin Jacob, Sahil Muhammed, Nithin R, Pankaj Kumar R, Edwin Thomas *National Institute of Technology Calicut*	88.99	150 s	
12	**Zhaojie, Zhipeng, Zhouxing & Qingyun** 🎓 Zhaojie Liu, Zhipeng Lv, Zhouxing Su, Qingyun Zhang *Huazhong University of Science and Technology*	87.90	1054 s	
13	**Marcin** 🎓 Marcin Mennemann *TU Dortmund*	75.06	30019 s	
14	**fu2025_pace** 🎓 Yi-Hui Fan, Arthur Gisbrecht, Rashid Harvey, Vinzent Jörß, Bernadette Keßler, Oliver Seitz *FU Berlin*	31.11	18421 s	
DQ	**fu2025_pace** 🎓 Yi-Hui Fan, Arthur Gisbrecht, Rashid Harvey, Vinzent Jörß, Bernadette Keßler, Oliver Seitz *FU Berlin*			

5.2.2 Short Description of the Dominating Set Heuristic Track Winning Solvers

Florian & Guillaume solve both Dominating Set and Hitting Set instances as Set Cover instances. In a first step they apply a set of reduction rules. They then compute greedily an initial solution, where a greedy heuristic is chosen depending on an inital very fast estimation of the solution size. Following up, a large neighborhood search is perfomed to very quickly improve the solution. Finally, a local search approach is used to improve the current solution to get the best possible solution.

Also **Root** unifies Dominating Set and Hitting Set by transforming them into Set Cover and applies a small set of classical reduction rules. It then generates a high-quality starting solution by applying a multi-round greedy heuristic guided by element frequencies. Finally, it iteratively improves the solution by local search with with an adaptive weighting approach.

Swats formulates both Dominating Set and Hitting Set as a generalized Dominating Set problem. It first applies a wealth of reduction rules, both known rules, as well as a variety of new reduction rules that were designed, optimized and implemented to be applicable even for very large graph instances. It then proceeds in an counter-example guided abstraction refinement (CEGAR) fashion. Iteratively it adds a subset of unsatisfied constraints and updates a solution to satisfy these constraints. This solution is found by a local search approach and subsequently improved by local search. If the size of the solution could not be improved for a few subsequent iterations, a state perturbation is applied to try to get out of a found local optimum.

5.3 Hitting Set – Exact Track

5.3.1 Ranking

Rank	Team	Score	Time	Link
1	**UzL** Max Bannach, Florian Chudigiewitsch, Marcel Wienöbst *European Space Agency, University of Lübeck*	79	49854 s	
2	**Bad Dominating Set Maker** Alexander Dobler, Simon D. Fink, Mathis Rocton *TU Wien*	79	57783 s	
3	**André** André Schidler *Albert-Ludwigs-Universität Freiburg*	78	49565 s	
4	**Alpaca** 🎓 Tobias Röhr *Hasso Plattner Institute, University of Potsdam, Germany*	76	50947 s	
5	**Shadoks** Guilherme D. da Fonseca, Fabien Feschet, Yan Gerard *Aix-Marseille Université, Université Clermont-Auvergne*	74	60729 s	
6	**Swats** Sylwester Swat *Poznań University of Technology*	64	67148 s	
7	**AEG Heidelberg** 🎓 Adil Chhabra, Marlon Dittes, Ernestine Großmann, Kenneth Langedal, Henrik Reinstädtler, Christian Schulz, Darren Strash, and Henning Woydt *Heidelberg University, Hamilton College*	63	77045 s	

5.3.2 Short Description of the Hitting Set Exact Track Winning Solvers

The **UzL** solver starts by applying four well-known (and fast) reduction rules before reformulating the problem as a (partial) Max-Sat problem. Afterwards, the Max-Sat solver `EvalMaxSat` [3] is used to solve the resulting Max-Sat instance. If the instance originates from the Dominating Set benchmark, an additional proprocessing step involving the preprocessor `maxpre2` [22] is done. The internal parameters of `EvalMaxSat` have been tweaked; in particular, the time spent on conflict minimization has been bounded. If the resulting Hitting Set instance is an instance of 2-Hitting Set (i.e., a Vertex Cover instance), similar approaches to the ones utilized in the PACE 2019 have been employed.

The **Bad Dominating Set Maker** first converts the input instance (both, Dominating Set and Hitting Set) into an equivalent Directed Constrained Domination instance. Afterwards, a large set of reduction rules is applied. Then, `hdt` [1] is used to compute a tree decomposition. If the resulting decomposition is of width at most 13, a dynamic programming approach is used to compute a solution. Otherwise, the solver checks whether the instance is a Vertex Cover instance, in which case the PACE 2019 solver `peaty` [30] is employed. If all the aforementioned condition do not hold, the Max-Sat solver `EvalMaxSat` [3] with some adjustments is used.

The solver of **André Schidler** first applies two well-known (and fast) reduction rules. Afterwards, based on the size of the instance, either a Max-Sat solver or a Mixed Linear Programm (MLP) solver is utilized: if the reduced instance has no more than 250 elements, the instances is directly translated into an MLP instances and solved using the MLP solver `SCIP` [6]. Otherwise a self-implemented Max-Sat solver is used, which relies on the OLL algorithm [27] and internally uses the Sat solver Cadical 2.1.3. [7]. The key to good OLL performance relies on a strong initial lower bound which is obtained by finding a large set of disjoint sets. To speed up things further, both, the MLP solver as well as the Max-Sat solver are warm-started with an upper bound which is obtained using a local search approach.

5.4 Hitting Set – Heuristic Track

5.4.1 Ranking

Rank	Team	Score	Time	Link
1	**Root** Canhui Luo, Zhipeng Lv, Zhouxing Su, Qingyun Zhang *Huazhong University of Science and Technology*	99.79	30024 s	
2	**Florian & Guillaume** Florian Fontan, Guillaume Verger	99.73	29535 s	
3	**Shadoks** Guilherme D. da Fonseca, Fabien Feschet, Yan Gerard *Aix-Marseille Université, Université Clermont-Auvergne*	99.21	31522 s	
4	**Swats** Sylwester Swat *Poznań University of Technology*	99.18	23089 s	

5	**AEG Heidelberg** Adil Chhabra, Marlon Dittes, Ernestine Großmann, Kenneth Langedal, Henrik Reinstädtler, Christian Schulz, Darren Strash, and Henning Woydt *Heidelberg University, Hamilton College*	98.01	30013 s	
6	**Greeduce** Adam Polak, Jonas Schmidt *Bocconi University*	95.73	30011 s	
7	**Deepak, Syed & Saurabh** Deepak Ajwani, Syed Mahmudul Kabir Ratul, Saurabh Ray *New York University Abu Dhabi, University College Dublin*	82.50	29464 s	
8	**Sebastian, Mirza, Patrick & Mariette** Sebastian Angrick, Mirza Redzic, Patrick Steil, Mariette Vasen *Karlsruhe Institute of Technology*	1.83	32055 s	

5.4.2 Short Description of the Hitting Set Heuristic Track Winning Solvers

Root and **Florian & Guillaume** reduce both Dominating Set and Hitting Set to Set Cover and used the same solver for both problems, see Section 5.2.2.

Shadoks applies a small set of standard reduction rules. It then ties to solve each connected component exactly with an MIP or MAXSAT solver, from small to large, and aborting after a while if no solution is found in due time. It then reduces the problem to MAXSAT and runs an anytime MAXSAT heuristic solver for around a minute. Finally, it iteratively uses a local search improvement until the time runs out.

6 Student Ranking

Besides the global ranking there is a student ranking, which is only eligible for submission where students are the main contributors, for example, teams of student being part of a student project. These student groups may be supervised by non-student people as long as the students are the main contributors. In each trach the three highest ranked student student teams are the winners of the student category.

7 Price Money

The price money of 4000€ was generously provided by Networks [28] and is distributed as follows.

- 1rst Price: 300€
- 2nd Price: 200€
- 3rd Price: 100€

- 1rst Student Price: 200€
- 2nd Student Price: 150€
- 3rd Student Price: 100€

8 PACE Organsition

The program committee of PACE 2025 consisted of Mario Grobler and Sebastian Siebertz, both from the University of Bremen. During the competition, the members of the steering committee were:

- (since 2023) Max Bannach (European Space Agency)
- (since 2023) Sebastian Berndt (Universität zu Lübeck)
- (since 2016) Holger Dell (Goethe University Frankfurt and IT University of Copenhagen)
- (since 2016) Bart M. P. Jansen (chair) (Eindhoven University of Technology)
- (since 2024) Philipp Kindermann (Universität Trier)
- (since 2021) André Nichterlein (Technical University of Berlin)
- (since 2022) Christian Schulz (Universität Heidelberg)
- (since 2024) Soeren Terziadis (TU Eindhoven)

The Program Committee of PACE 2026 will be chaired by Alexander Leonhardt, Manuel Penschuck, and Mathias Weller.

9 Conclusion

We would like to sincerely thank all participants of PACE 2025 for their remarkable contributions, creative approaches, and their patience in the face of occasional technical issues. We look forward to future editions that will continue to build bridges between theory and practice, and we are excited to see how innovative solvers and new ideas will further shape the field.

9.1 Lessons Learned

Organizing the 10th iteration of the PACE Challenge has also provided valuable insights. It turned out to be highly challenging to design a benchmark set that is both structurally rich and well balanced in difficulty; while some classes of instances were solved quickly by nearly all solvers, others (notably the Watts–Strogatz graphs) proved unexpectedly difficult. Closely related, although creating a timeline for the organization is straightforward, adhering to it in practice turned out to be considerably harder. The discrepancy between the public and private instance sets, again caused by the delayed inclusion of Watts–Strogatz graphs, understandably led to surprise and discussion among participants.

On the technical side, the process of setting up Docker containers for all solvers was very time-consuming, but the effort clearly paid off: it ensured portability, reproducibility, and a fair evaluation across the homogeneous infrastructure provided by the Data Science Center at the University of Bremen. Finally, the verification phase to ensure correctness of solvers proved to be extremely valuable. It revealed a range of issues, including small implementation bugs such as failure to handle comment lines in input files, conceptual oversights such as missing coverage of corner cases, and configuration problems such as insufficient numerical precision of external ILP solvers. Thanks to this process, nearly all issues could be resolved before the official evaluation, and in the end only two solvers had to be disqualified for producing incorrect or suboptimal solutions in the exact track. We thank Manuel Penschuck for preparing the STRIDE repository and for his valuable remarks in catching the aforementioned bugs!

9.2 Outlook

Despite their difficulty, the Watts–Strogatz instances also provided particularly valuable insights. In total, 14 instances in Dominating Set and 15 instances in Hitting Set could not be solved by any solver. At the same time, the best-performing solvers solved only 81 instances (DS) and 79 instances (HS), respectively. This implies that even the strongest solvers failed on at least five, resp. six, instances that some other solver was able to handle successfully. Thus, no single solver dominated across the entire instance set, and the comparison revealed genuine diversity in algorithmic strengths.

This observation opens a promising avenue for future research. Since different solvers excelled on different subsets of the benchmark, it becomes natural to ask whether the underlying algorithmic ideas and engineering concepts can be combined. A solver that unifies the complementary strengths of existing methods might be capable of solving all instances that were solved by at least one team.

The experiences from PACE 2025 reinforce the importance of combining strong theoretical foundations with robust engineering practices. Future iterations of the PACE Challenge will continue to build on these lessons, refining the evaluation methodology, strengthening the benchmarking process, and fostering collaboration across different areas of algorithm design. We are looking forward to PACE 2026!

References

1 Michael Abseher, Nysret Musliu, and Stefan Woltran. htd – A free, open-source framework for (customized) tree decompositions and beyond, 2017.

2 Jochen Alber, Hongbing Fan, Michael R. Fellows, Henning Fernau, Rolf Niedermeier, Fran Rosamond, and Ulrike Stege. A refined search tree technique for dominating set on planar graphs. *Journal of Computer and System Sciences*, 71(4):385–405, 2005. doi:10.1016/J.JCSS.2004.03.007.

3 Florent Avellaneda. A short description of the solver evalmaxsat, 2020.

4 Max Bannach and Sebastian Berndt. PACE Solver Description: The PACE 2023 Parameterized Algorithms and Computational Experiments Challenge: Twinwidth. In *18th International Symposium on Parameterized and Exact Computation (IPEC 2023)*, volume 285 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 35:1–35:14, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPICS.IPEC.2023.35.

5 Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

6 Ksenia Bestuzheva, Mathieu Besançon, Wei-Kun Chen, Antonia Chmiela, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Oliver Gaul, Gerald Gamrath, Ambros Gleixner, Leona Gottwald, Christoph Graczyk, Katrin Halbig, Alexander Hoen, Christopher Hojny, Rolf van der Hulst, Thorsten Koch, Marco Lübbecke, Stephen J. Maher, Frederic Matter, Erik Mühmer, Benjamin Müller, Marc E. Pfetsch, Daniel Rehfeldt, Steffan Schlein, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Boro Sofranac, Mark Turner, Stefan Vigerske, Fabian Wegscheider, Philipp Wellner, Dieter Weninger, and Jakob Witzig. The SCIP Optimization Suite 8.0. Technical report, Optimization Online, 2021. URL: http://www.optimization-online.org/DB_HTML/2021/12/8728.html.

7 Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froleyks, and Florian Pollitt. Cadical 2.0. In *Computer Aided Verification*, pages 133–152, Cham, 2024. Springer Nature Switzerland. doi:10.1007/978-3-031-65627-9_7.

8 Édouard Bonnet and Florian Sikora. The PACE 2018 Parameterized Algorithms and Computational Experiments Challenge: The Third Iteration. In *13th International Symposium on Parameterized and Exact Computation (IPEC 2018)*, volume 115 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:15, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPICS.IPEC.2018.26.

9 Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

10 IBM ILOG Cplex. V12. 1: User's manual for cplex. *International Business Machines Corporation*, 46(53):157, 2009.

11 Anuj Dawar and Stephan Kreutzer. Domination Problems in Nowhere-Dense Classes. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 4 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 157–168, Dagstuhl, Germany, 2009. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPICS.FSTTCS.2009.2315.

12 Holger Dell, Thore Husfeldt, Bart M. P. Jansen, Petteri Kaski, Christian Komusiewicz, and Frances A. Rosamond. The First Parameterized Algorithms and Computational Experiments Challenge. In *11th International Symposium on Parameterized and Exact Computation (IPEC 2016)*, volume 63 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:9, Dagstuhl, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPICS.IPEC.2016.30.

13 Holger Dell, Christian Komusiewicz, Nimrod Talmon, and Mathias Weller. The PACE 2017 Parameterized Algorithms and Computational Experiments Challenge: The Second Iteration. In *12th International Symposium on Parameterized and Exact Computation (IPEC 2017)*, volume 89 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:12, Dagstuhl, Germany, 2018. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPICS.IPEC.2017.30.

14 Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness I: basic results. *SIAM Journal on Computing*, 24(4):873–921, 1995. doi:10.1137/S0097539792228228.

15 M. Ayaz Dzulfikar, Johannes K. Fichte, and Markus Hecher. The PACE 2019 Parameterized Algorithms and Computational Experiments Challenge: The Fourth Iteration. In *14th International Symposium on Parameterized and Exact Computation (IPEC 2019)*, volume 148 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:23, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.IPEC.2019.25.

16 P. Erdös and A. Rényi. On random graphs i. *Publicationes Mathematicae Debrecen*, 6, 1959.

17 Henning Fernau. A top-down approach to search-trees: Improved algorithmics for 3-hitting set. *Algorithmica*, 57(1):97–118, 2010. doi:10.1007/S00453-008-9199-6.

18 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, 1979.

19 Ernestine Großmann, Tobias Heuer, Christian Schulz, and Darren Strash. The PACE 2022 Parameterized Algorithms and Computational Experiments Challenge: Directed Feedback Vertex Set. In *17th International Symposium on Parameterized and Exact Computation (IPEC 2022)*, volume 249 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:18, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPICS.IPEC.2022.26.

20 Petter Holme and Beom Jun Kim. Growing scale-free networks with tunable clustering. *Phys. Rev. E*, 65, 2002.

21 Holger H. Hoos and Thomas Stützle. Satlib: An online resource for research on sat. In *SAT2000*, pages 283–292. IOS Press, 2000. URL: www.satlib.org.

22 Hannes Ihalainen, Jeremias Berg, and Matti Järvisalo. Clause redundancy and preprocessing in maximum satisfiability. In *Automated Reasoning: 11th International Joint Conference, IJCAR 2022, Haifa, Israel, August 8–10, 2022, Proceedings*, pages 75–94. Springer-Verlag, 2022. doi:10.1007/978-3-031-10769-6_6.

23 Richard M. Karp. Reducibility among combinatorial problems. In R. E. Miller, J. W. Thatcher, and J. D. Bohlinger, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_9.

24 Leon Kellerhals, Tomohiro Koana, André Nichterlein, and Philipp Zschoche. The PACE 2021 Parameterized Algorithms and Computational Experiments Challenge: Cluster Editing. In *16th International Symposium on Parameterized and Exact Computation (IPEC 2021)*, volume 214 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:18, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPICS.IPEC.2021.26.

25 Philipp Kindermann, Fabian Klute, and Soeren Terziadis. The PACE 2024 Parameterized Algorithms and Computational Experiments Challenge: One-Sided Crossing Minimization. In *19th International Symposium on Parameterized and Exact Computation (IPEC 2024)*, volume 321 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:20, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPICS.IPEC.2024.26.

26 Lukasz Kowalik, Marcin Mucha, Wojciech Nadara, Marcin Pilipczuk, Manuel Sorge, and Piotr Wygocki. The PACE 2020 Parameterized Algorithms and Computational Experiments Challenge: Treedepth. In *15th International Symposium on Parameterized and Exact Computation (IPEC 2020)*, volume 180 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 37:1–37:18, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPICS.IPEC.2020.37.

27 Antonio Morgado, Carmine Dodaro, and Joao Marques-Silva. Core-guided maxsat with soft cardinality constraints. In *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Proceedings*, Lecture Notes in Computer Science, pages 564–573. Springer Verlag, 2014.

28 Networks Project, 2017. URL: https://www.thenetworkcenter.nl.

29 J.A. Telle and Y. Villanger. Fpt algorithms for domination in sparse graphs and beyond. *Theoretical Computer Science*, 770:62–68, 2019. doi:10.1016/J.TCS.2018.10.030.

30 James Trimble. jamestrimble/peaty: v1.0.0: Pace challenge 2019 entry, 2019.

31 Dekel Tsur. Faster parameterized algorithms for variants of 3-hitting set. *Journal of Combinatorial Optimization*, 49(4):1–14, 2025.

32 Johan M. M. van Rooij, Jesper Nederlof, and Thomas C. van Dijk. Inclusion/exclusion meets measure and conquer. In *Algorithms - ESA 2009*, pages 554–565. Springer Berlin Heidelberg, 2009. doi:10.1007/978-3-642-04128-0_50.

33 Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, 1998.

PACE Solver Description: OBLX Exact Solver for the Dominating Set Problem

Jona Dirks
Université Clermont-Auvergne, CNRS, Mines de Saint-Étienne, Clermont-Auvergne-INP, LIMOS, 63000 Clermont-Ferrand, France

Enna Gerhard
University of Bremen, Bibliothekstraße 5, 28359 Bremen, Germany

Victoria Kaial
Université Clermont-Auvergne, CNRS, Mines de Saint-Étienne, Clermont-Auvergne-INP, LIMOS, 63000 Clermont-Ferrand, France

Lucas Lorieau
Université Clermont-Auvergne, CNRS, Mines de Saint-Étienne, Clermont-Auvergne-INP, LIMOS, 63000 Clermont-Ferrand, France

Abstract

We present and describe the solver *OBLX* for the Dominating Set problem on graphs. This solver was developed during the PACE challenge 2025 for the Exact track. It first applies several data reduction rules and performs a polynomial time reduction to Max Sat. The resulting Max Sat instance is in turn solved using the *EvalMaxSat* solver by Florent Avellaneda.

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

Keywords and phrases complexity theory, parameterized complexity, linear programming, java, dominating set, PACE 2025

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.33

Supplementary Material *Software (Source Code)*: `https://gitlab.limos.fr/oblx/public` [4] archived at `swh:1:dir:2863e5f060c85b6d04b04c053ab39135abee8f0d`

Funding *Lucas Lorieau*: This author has received financial support from the CNRS through the MITI inter-disciplinary programs and the IRL ReLaX.

1 Introduction

We present a brief description of the solver *OBLX* submitted for PACE Challenge 2025 (`https://pacechallenge.org/2025/`). It is an exact solver for the Dominating Set problem, that is, given a graph select a set of vertices such that every vertex that is not in the set itself has a neighbor in it.

In Section 2, we present the notation and definitions that will be used in the following. We summarize the architecture of our solver in Section 3. Section 4 is dedicated to a description of the reduction rules and their correctness. Sketches of proofs are given for non-trivial rules.

The implementation and software design builds on the solver presented in Enna Gerhard's master thesis [6], which is an improved version of the PACE submissions [2] and [5].

2 Notation and preliminary definitions

We refer to [3] for common notation and terminology on graphs and parameterized complexity.

For our reduction rules, we define a new Extended Dominating Set problem, that is identical to the common version but differentiates between three types of vertices, each of them corresponding to a status with regards to a partial solution. The different considered

© Jona Dirks, Enna Gerhard, Victoria Kaial, and Lucas Lorieau;
licensed under Creative Commons License CC-BY 4.0

vertices' types are the following. *Covered vertices* are vertices that are dominated by the current partial solution built by reduction rules; *Excluded vertices* are vertices that are forbidden in any solution we will consider. This type of vertex is typically used when a reduction rule detects that a specific vertex won't be present in any optimal solution; *Plain vertices* are not covered and not excluded. An Extended Dominating Set instance containing only plain vertices is equivalent to a Dominating Set instance of the same vertices and edges.

This labeling of vertices based on the current partial solution is inspired by [7] where a similar partition is used for their own set of reduction rules. In contrast, we keep track of vertices added to the solution and remove them within the graph, which reduces the complexity of creating reduction rules.

3 Overview of the solving pipeline

Figure 1 presents the overview of the solving pipeline that we have created. We first iteratively apply a set of data reduction rules (see Section 4) in order to reduce the size of the instance to solve. We apply these rules exhaustively and always start again with the first rules once a rule has been applied successfully. When no reduction rule can be applied anymore, we can treat connected components independently. We perform two polynomial time reductions:

- We reduce our Extended Dominating Set instance to the Hitting Set problem on hypergraphs. In the new hypergraph, we use the same vertex set of our original graph. For each non-covered vertex of the original graph we create a new hyperedge for each closed neighborhood, connecting all the vertices of said neighborhood. This follows the well known straightforward classic reduction of the Dominating Set problem.
- We then encode our instance of Hitting Set as a Max Sat instance that will be finally solved using the *EvalMaxSat* solver [1]. We add soft constraints trying to exclude all vertices individually. For each hyperedge we create a hard constraint requiring it to be included in the solution.

The set of variables assigned as true in the solution of the Max Sat instance are exactly a solution to the Hitting Set instance, and in turn for the Extended Dominating Set instance that we had obtained after reduction rules. We now have to apply remaining instructions of reduction rules that depend on the solution of the reduced instance in backward order to obtain the solution for the original Dominating Set instance.

4 Description of the reduction rules

We present the different reduction rules used in the solver in Tables 1 and 2. We provide a quick description and justification of their correctness. The rules are applied iteratively: We attempt to apply the rules in their order, moving on the next rule if the rules pattern could not be matched. As soon as a rule has been applied successfully, we return to trying to applying rules beginning at the start. Simple rules that are easy to check are placed first in the order with expensive rules being applied at the end. After the last rule did not match, all reduction rules have been applied exhaustively.

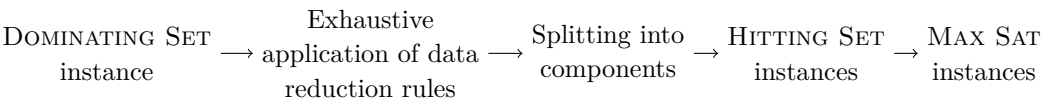


Figure 1 Solving pipeline.

Table 1 Data reduction rules (I).

Name and Description
1. Dominated covered Remove a covered vertex v if it has a non excluded neighbor that covers a superset of the closed neighborhood of v. Implied by Rule 2 of [7].
2. Isolated vertices Remove vertices v if $\deg(v) \leq 1$. Add v or its neighbor to the solution if necessary.
3. Exclude dominated Remove an excluded vertex v if a non-covered neighbor w exists that only adjacent to other neighbors of v. Any vertex selected to cover w also covers v.
4. All neighbors are excluded Directly include a non-covered vertex v if all of its neighbors are excluded.
5. Remove edges between excluded vertices (obviously possible)
6. Excluded covers Mark a vertex v as covered, if there exists an excluded vertex u such that $N(u) \subseteq N(v)$. If a vertex is adjacent to all neighbors of an excluded vertex, it will automatically be covered. This indirectly resolves domination between excluded vertices.
7. Dominated excluded Remove an excluded vertex v if it has a non-covered neighbor that needs to be covered by a subset of the neighborhood of v.
8. Diadems Let $v_1 \dots v_5$ be a chordless cycle such that (i) v_1, v_2 and v_3 are not covered and do not have any neighbors outside of the cycle, (ii) v_2, v_4 and v_5 are not excluded. Remove v_1, v_2 and v_3. Further add v_2 to the solution.
9. Remove edges between covered vertices (obviously possible)
10. Diamonds Assume $u_1, u_2, \dots, u_\ell$ ($\ell \geq 2$) are false twins with only two neighbors v_1, v_2 that are additionally required to not be excluded. Remove $u_2, \dots, u_\ell$ and exclude u_1. Note that any solution for the reduced graph is also a solution for the original graph. The size of an optimal solution does also not increase. Moving to a new solution of equal size: If a solution contains only one such vertex it has to be either v_1 or v_2 and we can keep that solution. Otherwise, we move to the solution containing v_1 and v_2 and none of the vertices $u_1, \dots u_\ell$.
11. Contract squares If there is a chordless cycle $v_1, \dots, v_4$ of non-covered non-excluded vertices with only v_1 and v_2 having outside neighbors and v_2 exactly one, u_2, then remove v_2, v_3, v_4 and add an edge v_1u_2, decreasing the solution size by one. For obtaining the original solution, we can add exactly one vertex: If $v_1 \in S$ then add v_2; if $u_2 \in S$ add v_4, otherwise add v_3, restoring the original coverage.
12. Ladders Let $v_{1,1}, \dots v_{3,2}$ be the vertices of an induced 3×2-grid without covered or excluded vertices. (a) If no vertices apart from $v_{1,2}$ and $v_{3,1}$ have neighbors outside of the grid, we add $v_{1,2}$ and $v_{3,1}$ to the solution. A minimum of two vertices are required to cover the grid, choosing the ones with (possible) external neighbors may cover additional vertices. (b) If no vertices besides $v_{1,1}$ and $v_{3,1}$ have neighbors outside of the grid, remove $v_{1,2}, v_{2,1}, v_{2,2}, v_{3,2}$ and add the edge $v_{1,1}v_{3,1}$. If both $v_{1,1}$ and $v_{3,1}$ are in S, add $v_{2,2}$ to the solution; if $v_{1,1} \in S$ but $v_{3,1} \notin S$, add $v_{3,2}$ to the solution; if $v_{3,1} \in S$ but $v_{1,1} \notin S$, add $v_{1,2}$ to the solution; and if both $v_{1,1}, v_{3,1} \notin S$, add $v_{2,2}$ to the solution. In all of the cases, the solution size decreases by one in the new reduced graph. Depending on $v_{1,1}$ and $v_{3,1}$ being covered in the solution in the original graph, a minimum solution would exactly contain the vertex we now add manually apart from the case where both are in the solution, in which an arbitrary additional vertex may be selected.

Table 2 Data reduction rules (II).

Name and Description
13. Small cat ears
If for two adjacent edges $\{v_1, v_2\}, \{v_2, v_3\}$, v_2 not covered or excluded, excluded vertices v_x, v_y with edges $\{v_x, v_1\}, \{v_x, v_2\}, \{v_y, v_2\}, \{v_y, v_3\}$ exist and v_2, v_x, v_y are otherwise isolated, we introduce a new covered vertex v_{13} and connect it to all vertices adjacent to v_1 or v_3. We then delete v_x, v_y, v_1, v_2, v_3. If v_{13} is included in the solution, we add both v_1, v_3, otherwise v_2. Since both v_x, v_y need to be covered, as soon as v_1 is chosen for the solution, we can push out a possibly selected v_2 to v_3 (the other way by symmetry), also covering the other side, otherwise both can be covered by v_2. This behavior is performed exactly by the newly created vertex.
14. Covered excluded sibling
If for two excluded vertices v_1, v_2, $N(v_1) \subseteq N(v_2)$, we can remove v_2 as it will be covered by every vertex used to cover v_1.
15. House
Special case of Rule 18 where $V(C) - m = \{v_1, v_2\}$, $N(m) = \{s, v_1, v_2\}$, $N(v_1) = \{s, t, m\}$ and $N(v_2) = \{m, t\}$.
16. Native ignorable vertices
More efficient implementation of a special case of Rule 3 of [7].
17. Extended native ignorable vertices
Implementation of Rule 3 of [7].
18. General s t
If there exists a connected component C of $G - \{s, t\}$ such that (i) $\{s, t\}$ dominate C, (ii) neither s nor t individually dominates C, and (iii) $\exists m \in V(C)$ that dominates C but not $C \cup \{s, t\}$, then we add a dominated vertex x with $N(x) = N(s) \cup N(t)$ and remove C, s and t. If for the new graph x is picked into the solution then pick m otherwise pick s and t. This works because in both cases the solutions size gets reduced by one and if s and t or m is in the solution C is dominated.

References

1 Florent Avellaneda. A short description of the solver EvalMaxSAT, 2020. URL: `https://hdl.handle.net/10138/318451`.

2 Moritz Bergenthal, Jona Dirks, Thorben Freese, Jakob Gahde, Enna Gerhard, Mario Grobler, and Sebastian Siebertz. PACE Solver Description: GraPA-JAVA. In *IPEC 2022*, volume 249 of *LIPIcs*, pages 30:1–30:4, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.IPEC.2022.30`.

3 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2016. `doi:10.1007/978-3-319-21275-3`.

4 Jona Dirks, Enna Gerhard, Victoria Kaial, and Lucas Lorieau. OBLX. Software, swhId: `swh:1:dir:2863e5f060c85b6d04b04c053ab39135abee8f0d` (visited on 2025-12-04). URL: `https://gitlab.limos.fr/oblx/public`, `doi:10.4230/artifacts.25239`.

5 Jona Dirks, Mario Grobler, Roman Rabinovich, Yannik Schnaubelt, Sebastian Siebertz, and Maximilian Sonneborn. PACE Solver Description: PACA-JAVA. In *IPEC 2021*, volume 214 of *LIPIcs*, pages 30:1–30:4, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.IPEC.2021.30`.

6 Enna Gerhard and Sebastian Siebertz. Solving the Directed Feedback Vertex Set Problem in Theory and Practice, 2024. `doi:10.26092/elib/4916`.

7 Ziliang Xiong and Mingyu Xiao. Exactly Solving Minimum Dominating Set and Its Generalization. *IJCAI-24*, 2024. `doi:10.24963/ijcai.2024/780`.

PACE Solver Description: Shadoks Approach to Minimum Hitting Set and Dominating Set

Guilherme D. da Fonseca
LIS, Aix-Marseille Université, France

Fabien Feschet
LIMOS, Université Clermont Auvergne, France

Yan Gerard
LIMOS, Université Clermont Auvergne, France

Abstract

Description of the solvers used by the Shadoks team in the PACE 2025 challenge. The challenge considers solvers for the minimum dominating set and hitting set problems. For the heuristic challenge, we respectively won third and fourth place for hitting set and dominating set. For the exact challenge, we won fifth place on both problems.

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

Keywords and phrases Optimization, heuristic, hitting set, dominating set

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.34

Supplementary Material
Software (Source Code): `https://github.com/gfonsecabr/shadoks-PACE2025`
archived at `swh:1:dir:1bab06742110e0d15d8feacc31101a047ae102f1`

Funding Work supported by the French ANR PRC grant ADDS (ANR-19-CE48-0005).

Acknowledgements We would like to thank the challenge organizers and other competitors for their time, feedback, and making this whole event possible.

1 Introduction

We describe the solvers used by the Shadoks team in the PACE 2025 challenge. The challenge considers solvers for the minimum dominating set and hitting set problems. Since it is easy to reduce minimum dominating set to minimum hitting set, we only consider the latter. We present both a heuristic and an exact algorithm for the problem, as our solution to both is linked in many aspects. For the heuristic challenge, we respectively won third and fourth place for hitting set and dominating set. For the exact challenge, we won fifth place on both problems.

Given a hypergraph $H = (V, \mathcal{E})$ with vertices V and hyperedges $\mathcal{E}$, a *hitting set* is a subset $S \subseteq V$ such that for all $E \in \mathcal{E}$ we have $S \cap E \neq \emptyset$. A *minimum hitting set* is a hitting set with the smallest number of elements. To reduce a dominating set instance graph $G = (V, E')$ to a hitting instance $H = (V, \mathcal{E})$, it suffices to set $\mathcal{E}$ as the set of closed neighborhoods of the vertices in V.

In Section 2, we describe the different building blocks of our solvers. In Section 3, we show how the heuristic and exact solvers put these building blocks together. The source code of our solver is available at `https://github.com/gfonsecabr/shadoks-PACE2025`.

© Guilherme D. da Fonseca, Fabien Feschet, and Yan Gerard;
licensed under Creative Commons License CC-BY 4.0

2 Tools and Techniques

Reduction Rules. There are three key reduction rules [6] that we apply.

1. If a hyperedge E is a superset of another hyperedge, then remove E.
2. If a vertex v hits a subset of another vertex, then remove v.
3. If a hyperedge has a single vertex v, then add v to the solution and remove all hyperedges that contain v.

For the large heuristic instances, we had to make the implementation of these reductions very efficient.

Greedy Heuristic. Greedy heuristics insert the vertex of maximum "value" in the solution at each step. The classic greedy heuristic sets the value of a vertex as the number of unhit hyperedges that it would hit. Since large hyperedges are likely to be hit anyway, we give a smaller weight to hitting large hyperedges. More precisely, given an estimate c of the cost of the solution (in our case, a previously found solution cost), the weight of a hyperedge E is $w(E) = \left(1 - \frac{c}{|V|}\right)^{|E|-1}$. The value of a vertex is the sum of the weights of the unhit hyperedges it hits.

Local Search. Local search replaces some vertices of the solution by a smaller (or equal sized) set of vertices, obtaining a new solution that is not larger than the previous one. We perform a large neighborhood local search by removing several vertices of the solution (that should not be too far from each other in the hypergraph), computing the hypergraph defined by the unhit hyperedges, reducing this hypergraph, and then solving it with a greedy heuristic or an exact algorithm.

Clique and Independent Set. We use the MCS [8] and MCQ [7] algorithms implemented in `https://github.com/darrenstrash/open-mcs`. Maximum cliques and independent sets appear in two places of our solver. The first one is inside EvalMaxSAT, where we replaced the original EvalMaxSAT maximum clique algorithm by MCQ for sparse graphs (density under 0.1) and MCS for dense graphs. There is a performance gain for the rare cases when EvalMaxSAT uses dense graphs, and a small performance loss otherwise, but the modern C++ implementation made it easier to code a time limit without risking memory leaks.

The second place where a maximum independent set solver is used is when the hitting set problem has all hyperedges of degree 2, which means we need to solve a minimum vertex cover problem. After performing the aforementioned reduction rules, there are 66 of the 200 public exact instances where the problems reduces to vertex cover. However, there are no instances where this happens among the 200 private exact instances.

Exact MaxSAT. It is easy to formulate minimum hitting set as a MaxSAT instance with one hard clause for each hyperedge and one variable (and soft clause) for each vertex. We used the 2024 version of EvalMaxSAT [2], which is based on the SAT solver Cadical [4]. We modified EvalMaxSAT in order to add a time limit to the computation. To make the task easier, we replaced the clique solver that EvalMaxSAT uses with the one mentioned above.

Heuristic MaxSAT. We also used the heuristic anytime MaxSAT solver tt-open-wbo-inc [5], especially the Nuwls solver in it [3]. This solver has not been modified and is called as a separate process with a timeout.

Integer Programming. Depending on the parameters used, EvalMaxSAT uses SCIP [1] for solving integer programming for around one minute before attempting to solve the problem directly. We noticed, however, that GLPK is significantly faster for small instances that can be solved in under half a second.

3 Our Solvers

Now that we have presented the main tools, we are ready to describe our solvers.

Heuristic Solver

1) We apply the three reduction rules exhaustively.
2) We try to solve each connected component exactly with Integer Programming (GLPK) or MaxSAT (EvalMaxSAT) solvers, from small to large, aborting if taking too long. In fact, the greedy heuristic is used too, as it is optimal if the solution size is at most 2. If all components are solved, then we are done.
3) We reduce the problem to MaxSAT and run an anytime MaxSAT heuristic solver for around one to two minutes.
4) We then improve the solution using large neighborhood local search until we run out of time.

Step 4 deserves a more detailed explanation. In this step, we use a routine that removes vertices from our solution and then, replaces the removed part of the solution by a new set with a smaller or equal number of vertices. After removing several vertices from the solution, we obtain a set of unhit hyperedges. This set of unhit hyperedges (and all vertices in them) defines a smaller hypergraph instance. We apply the reduction rules to the smaller hypergraph and then try to solve it exactly with GLPK or EvalMaxSAT.

We tried different strategies to choose which vertices to remove, with no significant difference in the result, as long as the removed vertices are close to each other in the hypergraph. In our final version, we use the following strategy. We keep a queue of vertices and a vector of hyperedges. The queue is initialized with a random vertex of the solution and the vector is initially empty. At each step, we extract a vertex v from the queue and remove it from the solution if it is present there. We then add all hyperedges that contain v to the vector, making sure that each hyperedge is added only once. If the vertex queue is empty, we choose a random hyperedge from the vector and add all its vertices (that have not been added before) to the vertex queue.

A crucial part is the size of the smaller hypergraph, which is controlled by a *limit* parameter. We make sure that we remove at least $limit/2$ vertices and get at least *limit* unhit hyperedges. Then, we use our main solver *i.e.* either EvalMaxSAT or GLPK on the smaller hypergraph. We regularly update the limit value as well as the main solver by independently testing GLPK and EvalMaxSAT on differnt values of *limit*. More precisely, we test three values *limit*, $limit + \delta$ and $limit - \delta$ to determine whether *limit* should be increased, decreased, or kept, so that the ratio between the gain and the computation time is maximized. We also increase *limit* if no improvement is obtained, and decrease it if the solver times out without a solution. Figure 1 shows the evolution of the solution size using different values of *limit*.

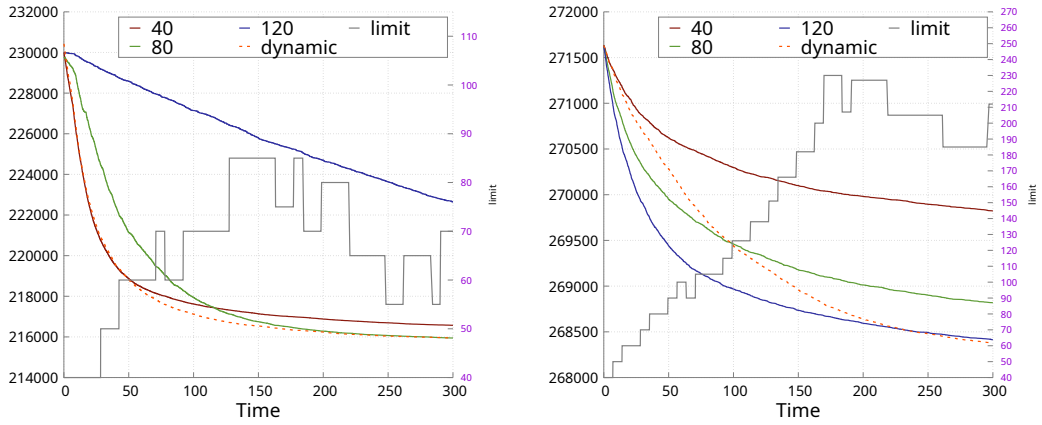


Figure 1 Size of the solution over a 5-minute time interval with different values of *limit* for the private hitting set instances 002 and 072 respectively. The dashed line shows the version that dynamically changes *limit* and the corresponding value of *limit* is represented by the gray curve (scale in purple on the right).

Exact Solver

1) We apply the three reduction rules exhaustively.

2) We try to solve each connected component exactly with Integer Programming (GLPK) or MaxSAT (EvalMaxSAT) solvers, from small to large, aborting if taking too long. If all components are solved, then we are done.

3a) If all hyperedges have degree 2 (and the number of vertices is at most 800), the problem is vertex cover and we solve it using the exact independent set solver MCS.

3b) Otherwise, we find a good heuristic solution (as steps 3 and 4 of the heuristic solver) and give it as an initial solution to EvalMaxSAT. Step 3b is repeated twice with different parameters.

References

1 Tobias Achterberg. SCIP: solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009. doi:10.1007/s12532-008-0001-1.

2 Florent Avellaneda. A short description of the solver EvalMaxSAT. *MaxSAT Evaluation*, 8:364, 2020.

3 Yi Chu, Shaowei Cai, and Chuan Luo. Nuwls: Improving local search for (weighted) partial maxsat by new weighting techniques. In *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023*, volume 2023, pages 3915–3923, 2023. doi:10.1609/aaai.v37i4.25505.

4 Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda. CaDiCal, Kissat, Paracooba, Plingeling and Treengeling entering the SAT competition 2020. *Artif. Intell.*, 301:103572, 2021. doi:10.1016/j.artint.2021.103572.

5 Alexander Nadel. Anytime algorithms for MaxSAT and beyond. In *2020 Formal Methods in Computer Aided Design (FMCAD)*, pages 1–1. IEEE, 2020. doi:10.34727/2020/isbn.978-3-85448-042-6_1.

6 Lei Shi and Xuan Cai. An exact fast algorithm for minimum hitting set. In *2010 Third International Joint Conference on Computational Science and Optimization*, volume 1, pages 64–67. IEEE, 2010.

7 Etsuji Tomita and Tomokazu Seki. An efficient branch-and-bound algorithm for finding a maximum clique. In *Discrete Mathematics and Theoretical Computer Science*, pages 278–289, 2003. doi:10.1007/3-540-45066-1_22.

8 Etsuji Tomita, Yoichi Sutani, Takanori Higashi, Shinya Takahashi, and Mitsuo Wakatsuki. A simple and faster branch-and-bound algorithm for finding a maximum clique. In *International workshop on algorithms and computation*, pages 191–203, 2010. doi:10.1007/978-3-642-11440-3_18.

PACE Solver Description: Bad Dominating Set Maker

Alexander Dobler
Algorithms and Complexity Group, TU Wien, Austria

Simon Dominik Fink
Algorithms and Complexity Group, TU Wien, Austria

Mathis Rocton
Algorithms and Complexity Group, TU Wien, Austria

Abstract

Dominating Set and Hitting Set are two well-known NP-hard problems on graphs and hypergraphs, respectively. For Dominating Set, we seek a subset S of vertices of minimum size, such that every vertex has a neighbor in S. For Hitting Set, we require that this minimum size subset S intersects each hyperedge. We present *Bad Dominating Set Maker*, our solver for both problems posed in the exact tracks of the 2025 PACE Challenge. It uses reduction rules, dynamic programming on tree decompositions, and external Vertex Cover and SAT solvers.

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

Keywords and phrases Dominating Set, Hitting Set, Pace Challenge

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.35

Supplementary Material *Software (source code)*: `https://github.com/Doblalex/pace2025` [3] archived at `swh:1:dir:77b0720cd14b92e8d17e114ccf5c1ea89f79dfc9`

Funding *Alexander Dobler*: Vienna Science and Technology Fund (WWTF) grant [10.47379/ICT19035].
Simon Dominik Fink: Vienna Science and Technology Fund (WWTF) grant [10.47379/ICT22029] and Austrian Science Fund (FWF) grant [10.55776/Y1329].
Mathis Rocton: European Union's Horizon 2020 research and innovation COFUND programme (LogiCS@TUWien, grant agreement No 101034440).

1 Introduction

Our algorithm can be broken down in five main steps. It first reads the input and constructs the corresponding instance of Directed Constrained Domination, which we define further down in this section. Note that only this step makes a difference between dominating set instances and hitting set instances. Once we have constructed the instance, we exhaustively apply a set of reduction rules described in Section 2. Some of these reduction rules split the instance into several smaller instances, in which case we solve these instances independently. After applying exhaustively the reduction rules, our algorithm uses several approaches, see Section 3. We use the library *htd* [1] to look for a tree decomposition of the undirected underlying graph with width smaller than 13. If we find such a decomposition we solve the problem using dynamic programming along the tree decomposition. Otherwise, if the instance corresponds to a Vertex Cover instance and has, informally, not too many candidates for the solution set, we run the dedicated solver *Peaty* [6] with a 5-minute time limit. If none of these specialized approaches succeed, we finally create a Max SAT instance equivalent to our current Directed Constrained Domination instance and use the *EvalMaxSAT* solver [2], with some improvements regarding the core computation.

© Alexander Dobler, Simon Dominik Fink, and Mathis Rocton;
licensed under Creative Commons License CC-BY 4.0

Directed Constrained Domination. We reduce both Dominating Set and Hitting Set to the following problem:

Problem Directed Constrained Domination
Input: A directed graph $G = (V, A)$, a *subsumed* set $S \subseteq V$ and a *dominated* set $D \subseteq V$.
Want: A set $DS(G) \subseteq V \setminus S$ of minimum size, such that $\forall v \in V \setminus D,\ \exists u \in DS(G), uv \in A$.

Note that A can contain arcs in both directions between two vertices, as well as self-loops. The transformation from Dominating Set to Directed Constrained Domination is trivial: the vertex set is the same, every edge is replaced by arcs in both directions, and no vertex is flagged as subsumed or dominated. Moreover, each vertex gets a self-loop.

For Hitting Set, create a dominated vertex for each element of the universe (i.e., the vertices of the hypergraph), and then a subsumed vertex for each set (i.e., each hyperedge). Add arcs from each dominated vertex to the subsumed vertices representing the sets it belongs to in the Hitting Set instance. Thus, any set will be hit if and only if the solution for Directed Constrained Domination contains a vertex belonging to this set. We have been made aware that a similar generalization of dominating set and some of our reduction rules have been presented in a recent IJCAI paper [10].

In the following, we assume any directed graph G to be implicitly accompanied by its labeling of subsumed and dominated vertices, and denote a solution to the corresponding Directed Constrained Domination problem as $DS(G)$. We use $[k]$ to denote $\{1, \ldots, k\}$ for $k \in \mathbb{N}$. For a vertex v or set of vertices X, we define $N_{\text{out}}(v)$ and $N_{\text{out}}(X)$ as its out-neighborhood and $N_{\text{in}}(v)$ and $N_{\text{in}}(X)$ as its in-neighborhood (including self-loops).

2 Reduction Rules

Our algorithm first preprocesses the instance with a set of reduction rules to simplify the instance and make it smaller, thus making it easier to handle for the exact solvers called later on. The reduction rules we use are adapted to our Directed Constrained Domination problem and in parts generalized from the rules available for Dominating Set in the work of Van Rooij and Bodlaender [7] and Volkmann [9].

The following rule always has to be applied first to ensure the safeness of all other rules.

▶ **Reduction Rule 1** (Arc removal). *If a vertex is dominated, remove all its in-arcs. If a vertex is subsumed, remove all its out-arcs.*

We adapt classical reduction rules on vertices with extreme degrees and connected components to the directed setting. For the rules 3 and above, mind that due to Reduction Rule 1, the existence of an uv arc implies that u is not subsumed and v is not dominated.

▶ **Reduction Rule 2** (Out-degree 0). *If there is a dominated vertex v with out-degree 0, remove v from the instance.*

▶ **Reduction Rule 3** (In-degree 1). *Let v be a vertex with a single incoming arc uv, and no outgoing arcs except for (possibly) vu. Add u to the dominating set and remove v.*

▶ **Reduction Rule 4** (Out-degree 1). *Let v be a dominated vertex with a single outgoing arc vu. If $u \notin S$ or has at least in-degree 2, remove v. Otherwise, add v to the dominating set.*

▶ **Reduction Rule 5** (Conn. components). *Connected components can be solved independently.*

▶ **Reduction Rule 6** (Subsumption). *Let u, v be two vertices. If $N_{out}(u) \supseteq N_{out}(v)$, mark v as subsumed. Independently, if $N_{in}(u) \subseteq N_{in}(v)$, mark v as dominated.*

The efficient implementation of Reduction Rule 6 requires an adapted *partition refinement* algorithm [5] to compute the set-inclusion of neighborhoods. Essentially, we implemented this algorithm with directed arcs between the different partitions which are based on the in/out-neighborhoods of the vertices.

▶ **Reduction Rule 7** (Contraction). *Let u, v be two vertices. If u is dominated but not subsumed, v is not dominated but subsumed, and there is an arc from u to v: mark u as not dominated, add to its in-neighborhood the in-neighborhood of v, and delete v.*

Proof. Observe that the corresponding Max Sat instances (see Section 3) before and after applying the reduction rules are exactly the same. ◀

The following two rules, following principles identified in rules 6 and 7 of [7], require more care when translating to our setting. Our first special rule is a generalization of rule number 6, considering that we do not have control over the maximal size of out-neighborhoods, and thus we need to compute a very local dominating set instead of simply counting elements. For efficient implementation, we want to apply the second special reduction rule several times without applying the subsumption reduction rule 6 between uses. Because of this, we need to make sure that some vertices do not have the same in-neighborhoods before reducing.

▶ **Reduction Rule 8** (Special 1 [7]). *Let u be a vertex and Q be the out-neighbors of u having in-degree 2. Let $P = N_{in}(Q)$. If $\exists p \in P \setminus \{u\} : N_{out}(P) = N_{out}(P \setminus p)$, add u to the solution.*

▶ **Reduction Rule 9** (Special 2 [7]). *Let v be a vertex with exactly 2 out-neighbors u_1, u_2 having in-degree 2 in G. Denote by w_1 and w_2 the remaining in-neighbors of u_1 and u_2, respectively. We proceed only if $w_1 \neq w_2$. Mark u_1 and u_2 as dominated, create a new dominated vertex w with out-neighborhood the union of the out-neighborhoods of w_1 and w_2, and subsume v, w_1, and w_2 to obtain instance G'. If $w \in DS(G')$, we have $DS(G) = (DS(G') \setminus w) \cup \{w_1, w_2\}$ otherwise we have $DS(G) = DS(G') \cup \{v\}$.*

▶ **Reduction Rule 10** (Cut-vertices [9]). *Let v be an (undirected) cut-vertex separating the subgraphs $B_1, \ldots, B_k$ in G, each including v. For $i \in [k]$, let B_i^- be the instance obtained from B_i by marking v dominated and let B_i^+ be B_i with $N_{out}[v]$ marked dominated. Then $DS(G)$ is the smallest of $\bigcup_i DS(B_i^+) \cup \{v\}$ and $\bigcup_{i \neq j} DS(B_i^-) \cup DS(B_j)$ for any $j \in [k]$.*

Note that, instead of applying this last reduction rule in one go, we instead iteratively apply it to individually remove leaf blocks of the block-cut-tree [4], and restrict this to blocks containing at most 25% of all vertices. This simplifies the implementation while avoiding multiple computations of a solution for large blocks, and also allows us to short-cut the computations in cases where we conclude that an optimal solution has already been found.

3 Exact Solvers and Other Improvements

Solving Low-Treewidth Instances. We use the *htd* library with the mindegree strategy to seek a nice tree-decomposition of small width [1]. If we obtain such a decomposition, we use a dynamic program to solve the instance in time $\mathcal{O}(3^{\text{tw}})$. To achieve this running time, our approach combines two different setups of the approach described in [8].

Namely, for join nodes, we consider tables where the state of the vertices are 1, $0_?$ and 0_0, whereas for introduce and forget nodes we consider the states 1, 0_1 and 0_0. 1 means the vertex is selected in the solution, 0_1 that it is not in the solution but neighbor to a vertex in the solution, 0_0 that it is neither in the solution nor has a neighbor in the solution. The additional $0_?$ state (which is introduced to make the table computation at join nodes faster) corresponds

to a vertex not selected in the solution, without information about its neighborhood. Our algorithm then translates the tables from one format to another according to the type of node we encounter in the tree decomposition. Additionally, we need to restrict the possible states for some vertices to adapt the algorithm to our needs: any subsumed vertex cannot be given state 1, and any dominated vertex can be given the state 0_1 (or $0_?$) for free.

Handling Vertex Cover Instances. We remark that some instances are encoding Vertex Cover in the following sense: if all the vertices with incoming arcs (i.e., the vertices needing to be dominated) have in-degree exactly 2, the instance corresponds to an instance of Vertex Cover where vertices with outgoing arcs are vertices, and vertices with ingoing arcs are edges between their two neighbors (a vertex can serve as an edge and a vertex simultaneously in this sense). For such instances with less than 1000 unsubsumed vertices, we run the exact solver *Peaty* for a maximum of 5 minutes [6].

EvalMaxSAT and Improvements. As final step we encode a remaining instance as SAT formula and use *EvalMaxSAT* to solve it [2]. The encoding is straight-forward, representing each vertex with a clause ensuring that at least one of its in-neighbors is selected. The selection of any vertex has a weight of -1, meaning as few vertices as possible are selected.

To improve the performance in our setting, we provide two adjustments. First, we use an adaptative time-out for the *core improvement* subroutine of the solver, to spend less time when we believe the lower bound to be already optimal. The second improvement takes care of providing an alternative initial lower bound to the solver: we look for a large matching in the accessory graph where there is an edge between two vertices if this pair is exactly the in-neighborhood of some vertex in the Directed Constrained Domination instance. Such a matching always provides an initial core for the Solver.

References

1 Michael Abseher, Nysret Musliu, and Stefan Woltran. htd - A free, open-source framework for (customized) tree decompositions and beyond. In *Proc. Integration of AI and OR Techniques in Constraint Programming (CPAIOR'2017)*, volume 10335 of *LNCS*, pages 376–386. Springer, 2017. https://github.com/mabseher/htd. doi:10.1007/978-3-319-59776-8_30.

2 Florent Avellaneda. Evalmaxsat 2023. *MaxSAT Evaluation 2023*, page 12, 2023. URL: https://github.com/FlorentAvellaneda/EvalMaxSAT.

3 Alexander Dobler, Simon Dominik Fink, and Mathis Rocton. The (not-so) Bad Dominating Set Maker. Software, Vienna Science and Technology Fund (WWTF) grant [10.47379/ICT19035], Vienna Science and Technology Fund (WWTF) grant [10.47379/ICT22029], European Union's Horizon 2020 research and innovation COFUND programme (LogiCS@TUWien, grant agreement No 101034440), Austrian Science Fund (FWF) grant [10.55776/Y1329], swhId: swh:1:dir:77b0720cd14b92e8d17e114ccf5c1ea89f79dfc9 (visited on 2025-12-05). URL: https://github.com/Doblalex/pace2025, doi:10.4230/artifacts.25245.

4 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.

5 Robert Paige and Robert Endre Tarjan. Three partition refinement algorithms. *SIAM J. Comput.*, 16(6):973–989, 1987. doi:10.1137/0216062.

6 Patrick Prosser and James Trimble. Peaty: an exact solver for the vertex cover problem, 2019. URL: https://github.com/jamestrimble/peaty.

7 Johan M. M. van Rooij and Hans L. Bodlaender. Exact algorithms for dominating set. *Discret. Appl. Math.*, 159(17):2147–2164, 2011. doi:10.1016/J.DAM.2011.07.001.

8 Johan M. M. van Rooij, Hans L. Bodlaender, Erik Jan van Leeuwen, Peter Rossmanith, and Martin Vatshelle. Fast dynamic programming on graph decompositions. *CoRR*, abs/1806.01667, 2018. arXiv:1806.01667.

9 Lutz Volkmann. A reduction principle concerning minimum dominating sets in graphs. *J. Comb. Math. Comb. Comp.*, 31:85–90, 1999.

10 Ziliang Xiong and Mingyu Xiao. Exactly solving minimum dominating set and its generalization. In *Proc. Conference on Artificial Intelligence, (IJCAI'2024)*, pages 7056–7064. ijcai.org, 2024. URL: https://www.ijcai.org/proceedings/2024/780.

PACE Solver Description: Reductions and Heuristic Search for the Dominating Set Problem and the Hitting Set Problem

Florian Fontan
Independent researcher, Grenoble, France

Guillaume Verger
Independent researcher, Lyon, France

Abstract

In this paper, we describe the solver we submitted for both heuristic tracks of the PACE challenge 2025 on the dominating set problem and the hitting set problem. We solve both problems as unicost set covering problems. Our solver first performs reductions on the instance. Then greedy algorithms generate an initial solution that serves as starting point of the large neighborhood search and the local search which are executed afterwards. The solver ranked first in the dominating set heuristic track, and second in the hitting set heuristic track.

2012 ACM Subject Classification Mathematics of computing → Combinatorial optimization

Keywords and phrases dominating set, hitting set, unicost set covering, reductions, large neighborhood search, local search

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.36

1 Outline

Both dominating set and hitting set instances are solved as unicost set covering instances.

The solving process follows four successive phases:

1. Instance reduction
2. Initial solution generated by greedy algorithms
3. Large neighborhood search
4. Local search

2 Instance reduction

The goal of this phase is to reduce the size of the instance to be solved.

The reductions applied are:

- Mandatory sets removal
- Dominated sets removal
- Dominated elements removal
- Set folding
- Unconfined sets reduction
- Small components reduction

The first three reduction rules are classical reductions for set covering problems. They correspond to Reduction 1 to 5 from [4].

The set folding reduction rule corresponds to Reduction 7 from [4] and is a generalization of the vertex folding reduction rule for the vertex cover problem. The idea is to replace a set j that covers two elements, together with its two neighboring sets j_1 and j_2, by a

© Florian Fontan and Guillaume Verger;
licensed under Creative Commons License CC-BY 4.0

single set j'. If j' is included in the solution of the reduced problem, then j_1 and j_2 must be included in the solution of the original problem; otherwise, j must be included in the solution of the original problem.

The unconfined reduction rule is a generalization of the unconfined reduction rule for the vertex cover problem [5]. The idea is to prob the inclusion of a set into an optimal solution and show that another optimal solution without this set exists. If this is the case, then the set can be removed from the instance.

For the small component reduction, we compute simple lower and upper bounds for each connected component of the instance. Any component for which the two bounds coincide can then be removed from the instance.

We also implemented generalizations of the twin reduction rule [5] and the crown reduction rule [3], but they did not have any effect on the challenge instances. Therefore, we did not enable them in our solver.

3 Initial solution

The large neighborhood search and the local search algorithms require an initial solution. Our initial solution generation procedure relies on 4 greedy algorithms.

- The classical greedy algorithm for the set covering problem: while not all elements are covered, the set covering the highest number of uncovered elements is added to the solution.
- The greedy algorithm from [2]: this greedy algorithm is similar to the classical greedy, but uses a different set scoring function.
- The dual greedy algorithm: in this greedy algorithm, while not all elements are covered, an uncovered element is selected, and a set covering it is added to the solution.
- The reverse greedy algorithm: this greedy algorithm starts with a solution containing all sets. Redundant sets are removed until none remain.

The first two greedy algorithms work well when the number of sets in the solution is small compared to the total number of sets. On the other hand, the reverse greedy algorithm works well when most sets are in the solution.

To determine which greedy algorithm to run, we assess the number of sets in the solution by running the dual greedy algorithm. The dual greedy algorithm is always very fast, although in general it yields worse solutions than the other greedy algorithms.

4 Large neighborhood search

The goal of this step is to quickly improve the solution obtained in the previous phase.

To this end, we apply a large neighborhood search algorithm. Each element is assigned a weight, initially weight of 1. The score of a set is then defined as the sum of the weights of the uncovered elements it covers.

The large neighborhood search works as follows. At each iteration:

- The set with the worst score is removed from the solution
- The score of the uncovered elements is incremented
- While not all elements are covered, the set with the best score is added to the solution and while the solution contains redundant sets, a redundant set is removed.

5 Local search

The goal of this last phase is to improve the current solution to get the best possible solution.

It implements a local search with a row weighting scheme similar to the one proposed by [2]. If the best feasible solution found so far contains k sets, the local search works on a solution containing $k-1$ sets. As a result, some elements remain uncovered. A weight is associated to each element. The penalty of a solution is defined as the sum of the weights of its uncovered elements. The goal of the local search is to find a solution with penalty 0. At each iteration, it performs a swap of minimal penalty between a set outside the solution that covers an uncovered element, and one of its set neighbor, that is, a set such that there exists an element that both sets cover. Thus, the solution still contains $k-1$ sets. Then the weight of each newly uncovered element is incremented.

When the number of set neighbors is too high so that computing them makes the solver hit the memory limit, it runs the local search with row weighting from [1] instead. Instead of looking for a swap, at each iteration, this local search first removes a set from the solution, and then adds another one that covers an uncovered element.

References

1 Chao Gao, Xin Yao, Thomas Weise, and Jinlong Li. An efficient local search heuristic with row weighting for the unicost set covering problem. *European Journal of Operational Research*, 246(3):750–761, November 2015. doi:10.1016/j.ejor.2015.05.038.

2 Weibo Lin, Fuda Ma, Zhouxing Su, Qingyun Zhang, Chumin Li, and Zhipeng Lü. Weighting-based parallel local search for optimal camera placement and unicost set covering. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, GECCO '20, pages 3–4, Cancún, Mexico, July 2020. Association for Computing Machinery. doi:10.1145/3377929.3398184.

3 William Suters, III. Crown Reductions and Decompositions: Theoretical Results and Practical Methods. *Masters Theses*, December 2004.

4 Johan M. M. van Rooij and Hans L. Bodlaender. Exact algorithms for dominating set. *Discrete Applied Mathematics*, 159(17):2147–2164, October 2011. doi:10.1016/j.dam.2011.07.001.

5 Mingyu Xiao and Hiroshi Nagamochi. Confining sets and avoiding bottleneck cases: A simple maximum independent set algorithm in degree-3 graphs. *Theoretical Computer Science*, 469:92–104, January 2013. doi:10.1016/j.tcs.2012.09.022.

PACE Solver Description: Minimum Hitting Set Computation via Core-Guided MaxSAT Solving

André Schidler
University of Freiburg, Germany

Abstract

This paper describes our hybrid MaxSAT and mixed integer programming approach for finding minimum hitting sets as submitted to the 2025 PACE challenge. We also discuss hitting set specific challenges, lower bounds, preprocessing and design choices.

2012 ACM Subject Classification Theory of computation → Discrete optimization; Theory of computation → Constraint and logic programming; Theory of computation → Automated reasoning

Keywords and phrases hitting set, maxsat, core-guided

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.37

Supplementary Material *Software*: `https://github.com/ASchidler/PACE25-submission` archived at `swh:1:dir:44546f4c17c6ee57221ccfa0ce05db41be0e310b`

Funding Funded by an Amazon Research Award (Fall/2023).

1 Introduction

The 2025 PACE Challenge evaluated (among others) new approaches for finding a *minimum hitting set (MHS)*, where given a set of sets $\mathcal{S}$ the goal is to find a minimum size set H – the hitting set – that intersects with all given sets, i.e., $\emptyset \neq H \cap S$, for all $S \in \mathcal{S}$. The decision version of the problem is one of the original 21 NP-complete problems identified by Karp [12]. We present the details to our approach that uses a combination of MaxSAT and mixed integer programming (MIP) to tackle the problem.

The use of a MaxSAT solver is motivated by the success of a MaxSAT approach for the directed feedback vertex set problem at the 2022 PACE challenge, which could be converted into an implicit hit set problem, where not all sets are known a priori [13, 14]. Further, MHS can be seen as a special case of MaxSAT, where variables occur in only one polarity. However, the MaxSAT solver solves only 94 of the 100 public competition instances within 30 minutes, which necessitates a complementary approach for the remaining instances.

MaxSAT solvers often use MIP solvers, either as a preprocessing step or to explicitly solve an (implicit) MHS posed by the MaxSAT solver [9, 16]. The use of a MIP solver before the actual MaxSAT algorithm is motivated by the observation that MIP solvers are often faster on small but hard instances [3], which matches the characteristics of the MHS instances not solved by our MaxSAT solver. While neither solver can solve all public competition instances, combined they are able to solve all 100 instances within 30 minutes.

2 Architecture

Our approach first preprocesses (Section 3) the instance and then passes it to either the MIP solver or our MaxSAT solver (Section 4) depending on the number of distinct elements of the instance: any instance with fewer than 250 distinct elements ($|\bigcup_{S\in\mathcal{S}} S|$) is passed on to the MIP solver.

© André Schidler;
licensed under Creative Commons License CC-BY 4.0

The preprocessing and MaxSAT implementations are ours and will be described subsequently. As the MIP solver, we use SCIP[1] [4] (version 8.1.0). Another advantage of this approach – although not used in our implementation – is that SCIP as well as MaxSAT offer a mode where the optimality of the solution can be certified [2, 8].

Correctness

The preprocessing, as subsequently discussed, is straightforward and not novel. We trust the correctness of the SCIP solver, as it is a well-established academic solver, based on proven methods. We implement the OLL [1, 15] algorithm for MaxSAT solving. The correctness of the algorithm has been proven in prior work and recently it has been shown that the algorithm, including all state-of-the-art extensions used in our solver, can be certified [2]. We only adjust the heuristics of the algorithm, hence, the approach remains correct.

3 Preprocessing

We use two simple preprocessing rules whose correctness follow from propositional logic and are also mentioned in hitting set literature [6].

1. Whenever there exist two distinct elements u and v such that v occurs in every set that u occurs in, we can remove u from all sets, as there exists a minimum hitting set not using u. Further, replacing u by v in a minimum hitting set yields another minimum hiting set.
2. Whenever there exist two distinct sets $S, S' \in \mathcal{S}$ such that $S \subset S'$, we can remove S', as any hitting set that intersects S also intersects S'.

Results show that the first rule is crucial for good performance, as SCIP does not solve several instances without this preprocessing. The reduction is also necessary to accurately identify instances that should be tackled using a MIP solver. Our experiments suggest that the second rule is unnecessary, as MIP and MaxSAT solvers already efficiently deal with this type of redundancy.

4 MaxSAT Solver

Proving that no hitting set of a given size exists is hard for SAT solvers when limiting the hitting set size with a single cardinality constraint. The reason for this behavior is that this propositional encoding provides very little structure that is exploitable by the SAT solver to avoid iterating over all possible sets of this given maximum size. Indeed, we expect that pure SAT encodings of the problem are similar to pigeonhole encodings, which are known to require exponential runtime for most SAT solvers [10]. This is relevant as most exact MaxSAT algorithms use a SAT solver in their backend.

We implement the core-guided algorithm OLL [1, 15] using Cadical 2.1.3 [5] with hyperparameters manually adjusted to MHS. Core-guided approaches use *cores* to dissect the MaxSAT instance into smaller parts and then connect these cores to form *meta-cores*. (Meta-)cores are used to reformulate the original MaxSAT instance, which introduces structure to the propositional formula visible to the internal SAT solver. This structure makes it easier for the SAT solver to find a minimum hitting set and prove optimality. Which (meta-)cores are used for these reformulations determines the structure of the propositional formula and can have immense impact on the time required to solve the instance [17].

[1] `https://www.scipopt.org`

Upper bounds are one way that can help find good (meta-)cors for MHS. We initially run local search for five seconds to get an initial solution. We use our implementation of the NuWLS local search algorithm [7] for this purpose. This upper bound guides the search for (meta-)cores, can avoid a costly last call to the SAT solver, and is also used to warmstart the MIP solver whenever the MHS instances is small.

Lower Bounds

We can initialize the MaxSAT solver with a promisingly structured reformulation using a good lower bound, a well-known method withing MaxSAT solvers [11]. The simplest lower bound is finding a large set of disjoint sets, where the cardinality of this set acts as a lower bound. This lower bound can be further improved whenever the instance contains sets of size two: let G be the graph created by viewing all sets of size two as edges. Then, a clique of size k implies that at least $k-1$ elements of the clique have to by in any minimum hitting set. This lower bound can be directly translated into OLL reformulations with a usually beneficial structure.

Finding good cliques is not the same as finding a large maximal clique: we can select several disjoint cliques but all cliques and other sets have to be disjoint. It is not established what the desired properties of such a lower bounding set are and different MaxSAT solvers follow different strategies. We repeatedly use a greedy heuristic using randomization and different criteria to find diverse sets of large cliques. For each such set of cliques, we greedily complete our lower bounding set adding other sets that are disjoint. We then choose the lower bounding set based on three criteria in descending priority: (i) high lower bound provided by the cliques, (ii) high lower bound provided by the whole set, and (iii) small cardinality of the lower bounding set.

5 Future Work

We use the MIP solver in our approach to solve small and complex instances. However, results from experiments running the MaxSAT solver repeatedly with different settings show that our solver can solve these instances. The problem is that the runtimes vary by orders of magnitude with several being within half an hour. Further analysis showed that this is mainly due to the structure introduced by the reformulations [17]. So far we were not able to establish a sufficiently good strategy for finding (meta-)cores that provides consistently good results, but see this as an interesting future challenge.

References

1 Benjamin Andres, Benjamin Kaufmann, Oliver Matheis, and Torsten Schaub. Unsatisfiability-based optimization in clasp. In *ICLP (Technical Communications)*, volume 17 of *LIPIcs*, pages 211–221. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPICS.ICLP.2012.211.

2 Jeremias Berg, Bart Bogaerts, Jakob Nordström, Andy Oertel, and Dieter Vandesande. Certified core-guided maxsat solving. In *Automated Deduction - CADE 29 - 29th International Conference on Automated Deduction, Rome, Italy, July 1-4, 2023, Proceedings*, volume 14132 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2023. doi:10.1007/978-3-031-38499-8_1.

3 Jeremias Berg, Matti Järvisalo, Ruben Martins, Andreas Niskanen, and Tobias Paxian. *MaxSAT Evaluation 2024: Solver and Benchmark Descriptions*. Department of Computer Science Series of Publications B. Department of Computer Science, University of Helsinki, Finland, 2024.

4 Ksenia Bestuzheva, Mathieu Besançon, Wei-Kun Chen, Antonia Chmiela, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Oliver Gaul, Gerald Gamrath, Ambros Gleixner, Leona Gottwald, Christoph Graczyk, Katrin Halbig, Alexander Hoen, Christopher Hojny, Rolf van der Hulst, Thorsten Koch, Marco Lübbecke, Stephen J. Maher, Frederic Matter, Erik Mühmer, Benjamin Müller, Marc E. Pfetsch, Daniel Rehfeldt, Steffan Schlein, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Boro Sofranac, Mark Turner, Stefan Vigerske, Fabian Wegscheider, Philipp Wellner, Dieter Weninger, and Jakob Witzig. The SCIP Optimization Suite 8.0. ZIB-Report 21-41, Zuse Institute Berlin, December 2021.

5 Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froleyks, and Florian Pollitt. CaDiCaL 2.0. In *CAV*, volume 14681 of *Lecture Notes in Computer Science*, pages 133–152. Springer, 2024. doi:10.1007/978-3-031-65627-9_7.

6 Thomas Bläsius, Tobias Friedrich, David Stangl, and Christopher Weyand. An efficient branch-and-bound solver for hitting set. In *Proceedings of the Symposium on Algorithm Engineering and Experiments, ALENEX 2022, Alexandria, VA, USA, January 9-10, 2022*, pages 209–220. SIAM, 2022. doi:10.1137/1.9781611977042.17.

7 Yi Chu, Shaowei Cai, and Chuan Luo. Nuwls: Improving local search for (weighted) partial maxsat by new weighting techniques. In *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023*, pages 3915–3923. AAAI Press, 2023. doi:10.1609/AAAI.V37I4.25505.

8 William J. Cook, Thorsten Koch, Daniel E. Steffy, and Kati Wolter. A hybrid branch-and-bound approach for exact rational mixed-integer programming. *Math. Program. Comput.*, 5(3):305–344, 2013. doi:10.1007/S12532-013-0055-6.

9 Jessica Davies and Fahiem Bacchus. Exploiting the power of MIP solvers in MaxSat. In *SAT*, volume 7962 of *Lecture Notes in Computer Science*, pages 166–181. Springer, 2013. doi:10.1007/978-3-642-39071-5_13.

10 Armin Haken. The intractability of resolution. *Theor. Comput. Sci.*, 39:297–308, 1985. doi:10.1016/0304-3975(85)90144-6.

11 Alexey Ignatiev, António Morgado, and João Marques-Silva. RC2: an efficient maxsat solver. *J. Satisf. Boolean Model. Comput.*, 11(1):53–64, 2019. doi:10.3233/SAT190116.

12 Richard M. Karp. Reducibility among combinatorial problems. In *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.

13 Rafael Kiesel and André Schidler. PACE solver description: Dager - cutting out cycles with maxsat. In *17th International Symposium on Parameterized and Exact Computation, IPEC 2022, September 7-9, 2022, Potsdam, Germany*, volume 249 of *LIPIcs*, pages 32:1–32:4. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.IPEC.2022.32.

14 Rafael Kiesel and André Schidler. A dynamic maxsat-based approach to directed feedback vertex sets. In *Proceedings of the Symposium on Algorithm Engineering and Experiments, ALENEX 2023, Florence, Italy, January 22-23, 2023*, pages 39–52. SIAM, 2023. doi:10.1137/1.9781611977561.CH4.

15 António Morgado, Carmine Dodaro, and João Marques-Silva. Core-guided MaxSAT with soft cardinality constraints. In *CP*, volume 8656 of *Lecture Notes in Computer Science*, pages 564–573. Springer, 2014. doi:10.1007/978-3-319-10428-7_41.

16 Paul Saikko, Jeremias Berg, and Matti Järvisalo. LMHS: A SAT-IP hybrid MaxSAT solver. In *SAT*, volume 9710 of *Lecture Notes in Computer Science*, pages 539–546. Springer, 2016. doi:10.1007/978-3-319-40970-2_34.

17 André Schidler and Stefan Szeider. Analyzing reformulation performance in core-guided maxsat solving. In *28th International Conference on Theory and Applications of Satisfiability Testing, SAT 2025, August 12-15, 2025, Glasgow, Scotland*, volume 341 of *LIPIcs*, pages 26:1–26:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.SAT.2025.26.

PACE Solver Description: HitS&DoSeS – Exact and Heuristic Solvers for the Dominating Set and Hitting Set Problems

Sylwester Swat
Institute of Computing Science, Poznan University of Technology, Poland

Abstract

This article briefly describes the most important algorithms and techniques used in HitS&DoSeS, a dominating set and hitting set solver submitted to the PACE 2025 contest (10th Parameterized Algorithms and Computational Experiments Challenge). Used approaches for the exact and heuristic tracks are described, for both the dominating set and the hitting set problems.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases dominating set, hitting set, exact algorithms, heuristic algorithms, large graphs, combinatorial optimization

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.38

Supplementary Material *Software*: `https://github.com/swacisko/pace-2025`
archived at `swh:1:dir:434f896b7cfd7ff42bfeeeaf03887aa2aba471e8`

Funding This research received funding from the National Science Centre, Poland (grant No. 2023/49/N/ST6/02506, PI: Sylwester Swat).

1 Problem description

1.1 Dominating set

In the dominating set problem, the goal is to find a smallest subset S of vertices of a given graph $G = (V, E)$ such that, for each node $v \in V$, the intersection $N[v] \cap S$ is nonempty.

1.2 Hitting set

In the hitting set problem, the goal is to find, for a given family F of sets over universe U, a smallest subset $S \subseteq U$ such that each set in F contains at least one element from S.

2 Solver overview

In this paper we provide a short description of the most important algorithms implemented in solver HitS&DoSeS. Due to a large variety of used methods, this description does not contain full information about the algorithms , their details and their behaviour in many distinct situations.

The HitS&DoSeS solver was designed to deal with the generalized dominating set problem (GDS), defined as follows. Given an undirected simple graph $G = (V, E)$ and sets of vertices $H \subseteq V$ and $X \subseteq V$, find the smallest subset $S \subset V \setminus X$ such that, for each node $v \in V \setminus H$, the intersection $N[v] \cap S$ is nonempty.

The solver implemented within HitS&DoSeS for the hitting set problem works by solving the generalized dominating set problem for a bipartite graph $G' = (V', E')$ with bipartition sets A, B and sets $H = A$, $X = B$, where nodes in set A represent elements from universe U, nodes in B represent sets from family F, and there is an edge $\{a, b\} \in E'$ if and only if an

© Sylwester Swat;
licensed under Creative Commons License CC-BY 4.0

element represented by a is contained in the set represented by b. This holds for both exact and heuristic solvers for the hitting set problem. Due to this, we will mainly focus on the description of the dominating set solver.

Many combinatorial optimization problems can be treated as an implicit hitting set problem. For example, in the directed feedback vertex set problem the goal is to find a set of vertices that hits all cycles in a given graph. In the GDS, however, the family of sets to hit is given explicitly. This fact may be treated both as an advantage and a disadvantage. On the one hand, the size of the family of sets to hit is clearly bounded (contrary to, e.g., the directed feedback vertex set, where the number of cycles in the graph can be exponential in the graph size). On the other hand, handling only a subset of all constraints can sometimes decrease the size of a problem to tackle and it usually brings some "natural noise" to the computation, which can be beneficial. In the heuristic solver implemented within HitS&DoSeS we simulate generation of sets to hit, as though they were not given explicitly. This enables us to easily add randomization and makes it easier to get out of local optima.

3 Preprocessing

We use an extensive set of data reduction rules for the dominating set problem. This set contains known rules for the dominating set problem ([1]), extensions of those rules, as well as a variety of new reduction rules that were designed, optimized and implemented to be applicable even for very large graph instances. For a given instance of the GDS, the goal is to find one of the following:

1. a node that belongs to some optimal solution,
2. a node that can be marked as hit (added to set H),
3. a node that can be marked as excluded (added to set X),
4. a modification in the graph structure that could make the graph smaller or the structure easier in some sense, usually measured as the size of a result obtained by the local search solver run on the reduced graph.

Designed rules are called exhaustively, until no rule can be applied further or for roughly 25% of total admissible running time, if the preprocessing did not terminate earlier. Many of our rules require lifting a solution after finding it for the reduced graph instance. This is usually due to the fact that it is often possible to conclude that there exists a solution that does not contain a node (the node can be added to X) or that a node can be marked as hit and added to H, but only if certain additional constraints hold. Many of these constraints are usually fairly difficult to be taken into account when finding a solution using distinct heuristics and approaches, but it is much easier to fix a solution obtained for the reduced instance to make it a valid one for the original instance.

4 Exact algorithm

The exact algorithm is conceptually very simple: reduce the graph as much as possible using data reduction rules, then solve the reduced instance using a "general-purpose" solver, applied to the hitting-set formulation of the dominating set problem. In HitS&DoSeS we tested the following algorithms to solve the hitting set problem:

1. Integer Linear Programming: HiGHS solver [7] or SCIP solver [4],
2. MaxSAT: solving using a MaxSAT approach used by the RC2 solver [8] and either CaDiCaL [3] or Glucose 3 [2] as a SAT solver,
3. Constraint Programming: CP-SAT solver [9] from OR-tools software [10].

We also considered a simple branch-and-reduce approach, but our proof-of-concept implementation proved to be much slower than the "general-purpose"-solver approach, hence its further development and optimization were abandoned. By default, the hitting set problem is solved using CP-SAT solver with three workers, as this proved to give the best results (compared to other approaches mentioned above).

5 Heuristic approach

The heuristic approach works in a *CEGAR (counter-example guided abstraction refinement [6])* fashion. Starting with an empty set C – a subset of a set $\mathbf{C}$ containing all neighborhood-constraints (sets of the form $N[v] \setminus X$) generated for the GDS instance, and an initially empty set S (a hitting set of sets in C), we iteratively add some subset of unsatisfied constraints (sets in $\mathbf{C}$ not hit by S), then update S to make it a valid hitting set of sets in C.

In order to generate a set of constraints to add to C, we select some number of nodes closest to S in the considered graph G, nodes farthest from S, nodes of smallest/largest degree and several randomly selected nodes; we then add the corresponding constraints. However, the total number of added constraints is limited to at most $\alpha \cdot |UH|^{\beta}$, where UH is the set of all nodes v for which the intersection $N[v] \setminus S$ is empty and α, β are parameters whose values depend on basic graph characteristics.

After selecting the set D of constraints to add, we find a hitting set S_D of all sets in D using a local search approach, then create a solution S' to $C \cup D$ by taking $S' = S \cup S_D$. Next we take $S = S'$ and $C = C \cup D$ and try to optimize S using a local search approach. If the size of S could not be improved for a few subsequent iterations, a state perturbation is applied to try to get out of a found local optimum. A state perturbation is achieved by removing some small random subset of nodes in S and/or sets in C (and fixing S afterwards).

6 Local search approach

Our local search solver for the hitting set problem works by, iteratively, removing a node from S and adding a node back to S, unless it is not necessary. A node to remove from S is preferable over other nodes if it has a smaller value of a scoring function $ca(v)$, where $ca(v)$ is equal to the number of sets in C that are hit only by v. In case of a tie, a bunch of other heuristics are used. A node to add to S is preferable over other nodes if it has a greater value of a scoring function $hu(v)$, where $hu(v)$ is equal to the number of sets in C that are unhit by S and contain v. In case of a tie, a bunch of other heuristics are used, which form variations of rules that can be found in several existing heuristic dominating-set solvers (see, e.g., [11]).

Among the approaches used for tie-breaking, we use the following:

1. tabu search to prohibit visiting the same state more than once,
2. configuration checking strategy [5],
3. selecting the less frequently chosen node if the two compared nodes do not have "similar" state-change frequencies,
4. selecting the node that did not change its state for the longest time, if those times the two compared nodes are not "similar".

To effficiently track changes to values necessary to select next node to add/remove we use a custom-made heap that allows efficient modification of those values without necessity of removing and reinserting the nodes from/to the set. Random node moves are also applied to increase variability. Such random moves usually consist in removing a node and some number of nodes closest to it from the current solution, then adding the same number of

nodes back but using the standard comparator to select nodes to add back. Another method of increasing variability used in HitS&DoSeS consists in "freezing" some nodes - prohibiting it from being selected for some number of consecutive iterations.

A variation to the standard add/remove approach is admitted and sometimes executed. This variation allows adding a node to S only if it will result in S being a valid solution. Since we start improvement with a valid solution, such a node always exists, but it might be the same that was just removed from S. We observed that this usually cuts the search space, but significantly improves the convergence speed for some instances.

7 Availability

The source code of HitS&DoSeS is freely available and can be found at `https://github.com/swacisko/pace-2025`.

References

1 Jochen Alber, Michael Fellows, and Rolf Niedermeier. Polynomial time data reduction for dominating set. *Journal of the ACM*, 51, May 2004. doi:10.1145/990308.990309.

2 Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern sat solvers. In *IJCAI'09: Proceedings of the 21st International Joint Conference on Artificial Intelligence*, pages 399–404, July 2009. URL: http://ijcai.org/Proceedings/09/Papers/074.pdf.

3 Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froleyks, and Florian Pollitt. CaDiCaL 2.0. In *Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part I*, volume 14681 of *Lecture Notes in Computer Science*, pages 133–152. Springer, 2024. doi:10.1007/978-3-031-65627-9_7.

4 Suresh Bolusani, Mathieu Besançon, Ksenia Bestuzheva, Antonia Chmiela, João Dionísio, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Mohammed Ghannam, Ambros Gleixner, Christoph Graczyk, Katrin Halbig, Ivo Hedtke, Alexander Hoen, Christopher Hojny, Rolf van der Hulst, Dominik Kamp, Thorsten Koch, Kevin Kofler, Jurgen Lentz, Julian Manns, Gioni Mexi, Erik Mühmer, Marc E. Pfetsch, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Mark Turner, Stefan Vigerske, Dieter Weninger, and Lixing Xu. The SCIP Optimization Suite 9.0. ZIB-Report 24-02-29, Zuse Institute Berlin, February 2024. URL: https://nbn-resolving.org/urn:nbn:de:0297-zib-95528.

5 Shaowei Cai and Kaile Su. Configuration checking with aspiration in local search for sat. *Proceedings of the AAAI Conference on Artificial Intelligence*, 26(1):434–440, September 2021. doi:10.1609/aaai.v26i1.8133.

6 Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement for symbolic model checking. *J. ACM*, 50(5):752–794, September 2003. doi:10.1145/876638.876643.

7 Q. Huangfu and Julian Hall. Parallelizing the dual revised simplex method. *Mathematical Programming Computation*, 10, March 2015. doi:10.1007/s12532-017-0130-5.

8 Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. Rc2: an efficient maxsat solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 11:53–64, September 2019. doi:10.3233/SAT190116.

9 Laurent Perron and Frédéric Didier. CP-SAT. URL: https://developers.google.com/optimization/cp/cp_solver/.

10 Laurent Perron and Vincent Furnon. OR-Tools. URL: https://developers.google.com/optimization/.

11 Enqiang Zhu, Yu Zhang, Shengzhi Wang, Darren Strash, and Chanjuan Liu. A dual-mode local search algorithm for solving the minimum dominating set problem. *CoRR*, July 2023. doi:10.48550/arXiv.2307.16815.

PACE Solver Description: UzL Solver for Dominating Set and Hitting Set

Max Bannach
European Space Agency, AI and Data Science Section, Noordwijk, The Netherlands

Florian Chudigiewitsch
Institute for Theoretical Computer Science, University of Lübeck, Germany

Marcel Wienöbst[1]
Institute for Theoretical Computer Science, University of Lübeck, Germany

Abstract

This document contains a short description of our solver for the dominating set and hitting set problems that we submitted to the exact tracks of the PACE Challenge 2025. The solver is based on a straightforward MaxSAT formulation supplemented by hitting-set-based reduction rules. It utilizes a clique solver if the reduced instance is a (small) input for the vertex cover problem and tries to match certain lower bounds by expressing the reduced instance as a SAT problem.

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

Keywords and phrases exact algorithms, dominating set, hitting set

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.39

Supplementary Material *Software*: `https://github.com/mwien/PACE2025`
archived at `swh:1:dir:17aaee5f0fbc12856d7a55455c66a5f21801d327`

1 Introduction

Dominating set and hitting set are classical NP-hard problems [6]. In the *dominating set problem* we seek, for a given undirected graph $G = (V, E)$, a minimum-size set of vertices S such that for every vertex $u \in V$, the vertex u itself or one of its neighbors is in S. In the *hitting set problem*, the task is to find in a given hypergraph $H = (V, E)$ a minimum-size set of vertices S such that every edge $\{u_1, u_2, \dots\} \in E$ has a non-empty intersection with S. Clearly, any dominating set instance can be expressed as a hitting set instance with $E = \{ N[u] \mid u \in V \}$, where $N[u]$ is the closed neighborhood of u.

The solver we describe can be used to tackle both problems. Internally, any dominating set input is directly translated to the corresponding hitting set instance as described above. Thus, from now on, we solely focus on the hitting set problem.

2 Presolving with Reduction Rules

To improve the solver performance and reduce the instance complexity, we apply four reduction rules for the hitting set problem in a presolving step. These rules are designed to simplify the hypergraph structure while preserving the optimal solutions:

[1] Corresponding author.

© Max Bannach, Florian Chudigiewitsch, and Marcel Wienöbst;
licensed under Creative Commons License CC-BY 4.0

1. If an edge consists only of a single vertex u, then add u to S and remove all edges that contain u.
2. If a vertex is in exactly one edge, and this edge has size at least two, then discard the vertex and remove it from the edge.
3. If an edge is a subset of another edge, keep only the smaller edge.
4. If there are two vertices u and v such that we have $u \in e$ for all edges e that contain v, then delete v.

The correctness of these rules is immediate: In Rule 1, the vertex u needs to be in any solution; in Rule 3, the discarded edge is hit by any solution; and for Rules 2 and 4, we can guarantee that there is always a solution without the discarded vertex by an exchange argument. For an efficient implementation, we store and update not only the edges E but also, for each vertex, the indices of the edges the vertex is part of (and pointers to its position in these edges). To check whether an edge e is a subset of another edge f, we only consider edges f containing the vertex $u \in e$ that is in the fewest number of edges. We use the same optimization when applying Rule 4.

Reduction Rules for the Dominating Set Track

It is straightforward to see that the four hitting set reduction rules presented above subsume several well-known reduction rules for the dominating set problem – for example, the so-called "Rule 1" introduced by Alber, Fellows, and Niedermeier [1]. This justifies our approach in the dominating set track, to directly reduce the problem to hitting set and then apply the aforementioned hitting set reduction rules, without relying on any dominating set-specific reductions at all.

Performance of the Reduction Rules in the Competition

For the instances in the PACE Challenge 2025, our presolving step executes in fractions of a second and significantly reduces the size of the hypergraphs. While this leads to faster solving times for many instances, we observed that, on the private test set, the number of instances solvable within 30 minutes did not increase.

3 MaxSAT Formulation of Hitting Set

The hitting set problem can be naturally expressed as the (partial) MAX-SAT problem [4]. This problem allows for both *soft* clauses and *hard* clauses, where the hard clauses must all be satisfied and the goal is to satisfy as many soft clauses as possible. Let $H = (V, E)$ be a hypergraph, then we formulate the hitting set problem as

Soft Clauses $\bigwedge_{u \in V} (\neg u)$,

Hard Clauses $\bigwedge_{\{u_1, u_2, \dots\} \in E} (u_1 \vee u_2 \vee \cdots)$.

We solve this formulation using the EvalMaxSAT solver [2]. For the dominating set track, we additionally preprocess the formula using maxpre2 [7, 9]. We do not perform this preprocessing for the hitting set instances because it did not yield performance gains on the public instances and was counterproductive in some cases. On the private instances, we observed no clear improvements due to the use of maxpre2.

Furthermore, we tweaked internal parameters of EvalMaxSAT to the problems at hand and, notably, bounded the time spent in the conflict minimization phase depending on the conflict size. One of the effects of this parameter tuning is that many easier instances are solved significantly faster compared to using the off-the-shelf EvalMaxSAT solver. On harder instances, the effect is less significant as the SAT solving dominates the computation time.

4 Using Maximum Clique and SAT Solvers

In the case that the reduced instance is a *vertex cover* instance (i.e., $|e| \leq 2$ for all $e \in E$) and has at most 350 vertices, we solve the problem in a different way. Inspired by the results of the PACE 2019 [5], we use a maximum clique solver, namely mcqd [8], on the complement graph of H. Every vertex not in the maximum clique must be in a minimum-size vertex cover. Note that we do not use this strategy for instances with more than 350 vertices, where the size of the usually very dense complement graph could be prohibitively large. Here, we use the MAX-SAT formulation stated above.

Before launching the MAX-SAT solver, we attempt to compute a strong lower bound that can potentially be matched using a SAT formulation. Specifically, we greedily search for a perfect matching in the hypergraph – i.e., we look for $|V|/2$ hyperedges of size two such that each vertex appears in exactly one of these edges. If such a matching is found, any feasible solution S must have size at least $|V|/2$. In this case, we use the SAT solver kissat [3] to verify whether this lower bound can be matched by a solution that selects exactly one vertex per matching edge. Our encoding simply introduces a Boolean variable for each matching edge, representing which of its two vertices is included in the solution. We then add clauses for all remaining hyperedges to ensure they are also hit by the selected vertices.

While the vertex cover and the matching trick seemed to be promising during preparation, we observed after the competition that they did not trigger on any of the private instances of the PACE 2025 exact tracks. Thus, they did not affect the results of the competition.

5 Sketch of Correctness

For a brief sketch of a correctness proof, observe that the reduction rules are sound and the MAX-SAT formulation for the hitting set problem is immediate (and well-known). Solving vertex cover with a clique solver in the complement graph is a textbook trick. If the SAT solver finds an assignment of size $|V|/2$ in the instances discussed above, then clearly this solution is an optimal hitting set as it matches the lower bound.

As external solvers, we rely on well-established and open-source software: EvalMaxSAT [2], kissat [3], maxpre2 [7, 9], and mcqd [8].

References

1 Jochen Alber, Michael R Fellows, and Rolf Niedermeier. Polynomial-time data reduction for dominating set. *Journal of the ACM (JACM)*, 51(3):363–384, 2004. doi:10.1145/990308.990309.

2 Florent Avellaneda. A short description of the solver EvalMaxSAT. *MaxSAT Evaluation*, 8:364, 2020.

3 Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froleyks, and Florian Pollitt. CaDiCaL, Gimsatul, IsaSAT and Kissat entering the SAT Competition 2024. In *Proceedings of the SAT Competition 2024 – Solver, Benchmark and Proof Checker Descriptions*, pages 8–10, 2024.

4 Armin Biere, Marijn Heule, and Hans van Maaren. *Handbook of satisfiability*, volume 185. IOS press, 2009.

5 M. Ayaz Dzulfikar, Johannes K. Fichte, and Markus Hecher. The PACE 2019 Parameterized Algorithms and Computational Experiments Challenge: The Fourth Iteration. In *14th International Symposium on Parameterized and Exact Computation (IPEC 2019)*, pages 25:1–25:23, 2019. doi:10.4230/LIPIcs.IPEC.2019.25.

6 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

7 Hannes Ihalainen, Jeremias Berg, and Matti Järvisalo. Clause redundancy and preprocessing in maximum satisfiability. In *Proceedings of the 11th International Joint Conference on Automated Reasoning (IJCAR '22)*, volume 13385 of *Lecture Notes in Computer Science*, pages 75–94. Springer, August 2022. doi:10.1007/978-3-031-10769-6_6.

8 Janez Konc and Dušanka Janezic. An improved branch and bound algorithm for the maximum clique problem. *proteins*, 4(5):590–596, 2007.

9 Tuukka Korhonen, Jeremias Berg, Paul Saikko, and Matti Järvisalo. Clause redundancy and preprocessing in maximum satisfiability. In *Proceedings of the 20th International Conference on Theory and Applications of Satisfiability Testing, (SAT '17)*, volume 10491 of *Lecture Notes in Computer Science*, pages 449–456. Springer, 2017.

PACE Solver Description: Weighting-Based Local Search Heuristic for the Hitting Set Problem

Canhui Luo
Huazhong University of Science and Technology, Wuhan, China

Qingyun Zhang
Huazhong University of Science and Technology, Wuhan, China

Zhouxing Su[1]
Huazhong University of Science and Technology, Wuhan, China

Zhipeng Lü
Huazhong University of Science and Technology, Wuhan, China

Abstract

We present a unified heuristic solver for the PACE 2025 challenge, addressing both the dominating set and hitting set problems by reducing them to the unicost set covering problem. Our solver applies standard reduction rules, a multi-round frequency-based greedy initializer, and a local search guided by adaptive element weights. Additional techniques, such as component-level exact solving and swap restriction, further enhance performance. In the final official evaluation, our proposed solver achieved second place in the heuristic track for the dominating set problem of the PACE 2025 challenge, while securing first place in the heuristic track for the hitting set problem.

2012 ACM Subject Classification Mathematics of computing → Combinatorial optimization; Computing methodologies → Search methodologies

Keywords and phrases PACE 2025, Dominating Set, Hitting Set, Heuristic Optimization, Weighted Local Search

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.40

Supplementary Material *Software (Source Code)*: `https://github.com/lxily/PACE2025.DS-HS` [2] archived at `swh:1:dir:ae6eed6e943ad358e44567283df25ea7123a5fc7`

Funding This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grant 62402191, the Interdisciplinary Research Program of Hust under Grant 5003300129, and Innovation Program for Quantum Science and Technology under Grant No. 2024ZD0300500.

Acknowledgements We want to thank the organizers of PACE 2025 challenge and all other participants for creating such an engaging challenge.

1 Challenge Problem and Transformation

The PACE 2025 challenge involves two fundamental NP-hard problems: the dominating set problem and the hitting set problem. The dominating set problem seeks to select as few vertices as possible in a given graph such that every other vertex is adjacent to at least one selected vertex. The hitting set problem requires selecting as few elements as possible from a set system such that each set contains at least one selected element. In fact, the former can be regarded as a special case of the latter. We unify these two problems by transforming them into a unicost set covering problem, as follows:

[1] Corresponding author: Zhouxing Su

© Canhui Luo, Qingyun Zhang, Zhouxing Su, and Zhipeng Lü;
licensed under Creative Commons License CC-BY 4.0

- In the dominating set problem, each vertex v_i is represented as both a set s_i and an element e_i. Each set s_i covers all elements corresponding to its neighboring vertices.
- In the hitting set problem, each original set s_i^o is mapped to an element e_i, and each original element $e_j^o \in s_i^o$ is mapped to a set s_j covering e_i.

Subsequently, the optimization objectives of the two original problems are unified as minimizing the number of selected sets that cover all elements, although such a transformation inevitably loses certain characteristic information of the original problems.

2 Solver Methodology

In this section, we present our heuristic solver for the set covering problem. It integrates instance reduction, initialization, element-weighted local search, and two simple but effective optimization strategies that further enhance the search process.

2.1 Preprocessing via Reduction Rules

Considering that the datasets in the PACE 2025 Challenge may contain millions of sets and elements, applying possible reductions to them is highly beneficial for the subsequent search. We employ three classical reductions without sacrificing optimality [3]:

- **Element Dominance:** If all sets that cover element e_i also cover element e_j (where $i \neq j$), then e_j can be safely removed.
- **Set Dominance:** If all elements covered by set s_i are also covered by set s_j (where $i \neq j$), then s_i can be safely removed.
- **Mandatory Coverage:** If an element e_i is uniquely covered by one set s_i, then s_i must be selected, and all elements covered by s_i can be removed.

These reduction rules preserve the existence and structure of optimal solutions. By iteratively applying them, the problem size can typically be reduced significantly. We solve the reduced instance and merge the optimized result with the sets fixed during the reduction process to form the final solution.

2.2 Frequency-Guided Initialization

We define the *importance score* $\mathrm{IS}(e_i) = \frac{1}{\mathrm{freq}(e_i)}$ of an element as the reciprocal of its frequency (i.e., the number of sets that can cover it). Then, the score of a set $\mathrm{IS}(s_i) = \sum_{e_j \in \mathrm{uncovered}(s_i)} \mathrm{IS}(e_j)$ is the sum of the scores of all currently uncovered elements it covers. The greedy algorithm iteratively selects the set with the highest score, thereby prioritizing the coverage of hard-to-cover elements. After each selection, scores of related sets are updated to reflect the new uncovered element set.

Since the number of sets that can cover an element differs from the number of sets that actually cover it in a solution, the original importance score may be biased. To correct this, after obtaining a feasible solution, we refine the score of each element e_i by multiplying it with a scaling factor, i.e., $\mathrm{IS}'(e_i) = \mathrm{IS}(e_i) \cdot \frac{c_{\max}}{c_i}$, where c_i is the number of times element e_i is actually covered in the solution, and $c_{\max}$ is the maximum coverage count among all elements. Through multiple rounds of score refinement and reconstruction, higher-quality initial solutions can typically be obtained, albeit with increased construction time.

2.3 Element-Weighted Local Search

Starting from a feasible initial cover, we iteratively remove a randomly selected set and attempt to reconstruct a feasible cover without increasing the number of sets used. Once such a reconstruction is successful, we obtain an improved solution. Thus, our optimization focuses on how to achieve full coverage using a fixed number of sets.

2.3.1 Weighting Technique

Weighting techniques have demonstrated strong effectiveness in various set covering–related problems [3, 4, 5]. Our solver adopts a similar strategy: we assign weights to currently uncovered elements, and seek to minimize the total weight of uncovered elements during local search. Compared to minimizing just the count of uncovered elements, the weighted objective yields a smoother search landscape and better optimization performance.

2.3.2 Neighborhood Search

The neighborhood is defined by a pairwise swap operation: removing one set from the current solution and adding another. In each iteration:

- Randomly select an uncovered element e such that the subsequent swap operation ensures it will be covered. This random selection strategy enhances the diversification of the search while simultaneously reducing the evaluation time complexity.
- For each element e, we consider adding a set that covers it and simultaneously removing one set from the current solution. We evaluate all swap pairs and select the one that minimizes the total weight of uncovered elements after the move.

When the search reaches a local optimum, we increase the weight of a random uncovered element, encouraging its coverage in future iterations. Tabu search is a well-known metaheuristic for combinatorial optimization [1]. We integrate a one-iteration recency-based tabu mechanism that temporarily forbids recently involved sets from participating in swaps, thus encouraging search diversification and preventing cycling. Upon finding a new feasible solution (i.e., all elements are covered), we proactively remove a random set, and the optimization moves to the new bottleneck of one fewer set. The search procedure is repeated until the time limit is reached and the best solution found so far is returned.

2.4 Additional Enhancements

For instances that contain multiple connected components after reduction, we introduce two dedicated optimization strategies:

- Reducing the number of connected components: We apply a simple branch-and-bound algorithm to exactly solve connected components that involve fewer than k sets (with $k = 23$ in our implementation).
- Restricting active components: Initially, the removal of a set introduces uncovered elements in only one connected component. In subsequent swap iterations, if the added set fails to restore full coverage within this component while the removed set belongs to a different component, we revert the current component to its last feasible state.

References

1 Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, 13(5):533–549, 1986. doi:10.1016/0305-0548(86)90048-1.

2 Canhui Luo. lxily/PACE2025.DS-HS. Software, swhId: swh:1:dir:ae6eed6e943ad358e44567283df25ea7123a5fc7 (visited on 2025-12-03). URL: https://github.com/lxily/PACE2025.DS-HS, doi:10.4230/artifacts.25227.

3 Zhouxing Su, Qingyun Zhang, Zhipeng Lü, Chu-Min Li, Weibo Lin, and Fuda Ma. Weighting-based variable neighborhood search for optimal camera placement. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 12400–12408, 2021. doi:10.1609/AAAI.V35I14.17471.

4 Qingyun Zhang, Zhipeng Lü, Zhouxing Su, and Chumin Li. A vertex weighting-based double-tabu search algorithm for the classical p-center problem. *Computers & Operations Research*, 160:106373, 2023. doi:10.1016/J.COR.2023.106373.

5 Qingyun Zhang, Zhipeng Lü, Zhouxing Su, Chumin Li, Yuan Fang, and Fuda Ma. Vertex weighting-based tabu search for p-center problem. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pages 1481–1487, 2021.